ARBEITSBERICHTE
DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

HERAUSGEBER: H. Billing, W. Händler, U. Herzog, F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees, H. Niemann, D. Seitzer, H.J. Schneider

Zur ingenieurwissenschaftlichen Theorie von Rechensystemen

Alfred Lothar Luft

DISSERTATION

BAND 12 · NUMMER 4 · ERLANGEN · AUGUST 1979

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41546219_0033

ARBEITSBERICHTE
DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

HERAUSGEBER: H. Billing, W. Händler, U. Herzog, F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees, H. Niemann, D. Seitzer, H. J. Schneider

Zur ingenieurwissenschaftlichen Theorie von Rechensystemen

Dem Fachbereich Ingenieurwissenschaften
der Universität Erlangen-Nürnberg
zur Erlangung des Grades

DOKTOR - INGENIEUR

vorgelegt von

Alfred Lothar Luft

Erlangen, 1979

DISSERTATION

BAND 12 · NUMMER 4 · ERLANGEN · AUGUST 1979

UNIV.
LEIDEN
BIBL.

Als Dissertation genehmigt vom
Fachbereich Ingenieurwissenschaften
der Universität Erlangen-Nürnberg

Tag der Einreichung: 28. Mai 1979

Tag der Promotion: 24. Juli 1979

Dekan: Prof. Dr. D. Seitzer

Berichterstatter: Prof. Dr. H. Müller

Prof. Dr. H.J. Schneider

A b s t r a c t

Luft, Alfred Lothar:
ZUR INGENIEURWISSENSCHAFTLICHEN THEORIE VON RECHENSYSTEMEN

Für den Entwurf von-Neumann-ähnlicher Rechensysteme wird eine Schnittstelle zwischen anwenderorientierter Systembeschreibung und technologischer Realisierung entwickelt, die klar zwischen anwenderinvarianten und anwenderspezifischen Systemkomponenten unterscheidet und aus einer algorithmischen und einer architektonischen Modellbildung besteht. Beiden Modellbildungen liegt eine auf Grundbegriffen der formalen Logik aufbauende Präzisierung des Begriffs Anwenderbereich zugrunde. Die algorithmische Modellbildung beschreibt unter Bezug auf eine jeweils fest vorgegebene Menge von Anwenderbereichen ein Programmsystem, das aus Wertzuweisungen, standardisierten Verzweigungen und Schleifen, hierarchisch verknüpften lokalen und globalen Prozeduren sowie aus parallelen Klauseln bestehen kann. Die architektonische Modellbildung beschreibt für ein jeweils fest vorgegebenes Programmsystem mithilfe abstrakter Prozessor-, Speicher- und Bus-Bausteine eine "geeignete" Rechnerarchitektur, die sowohl einer realisierungsbezogenen Widerspiegelung der Semantik des Programmsystems als auch der Darstellung anwenderspezifischer Durchsatzforderungen dient. Die Programmsysteme und die Rechnerarchitekturen sind modular aufgebaut, einheitlich strukturiert sowie leicht erweiterungsfähig. Die Rechnerarchitekturen vermeiden Nachteile des monolithischen "universellen" von-Neumann-Rechners.
Die Frage nach der Korrektheit von Programmsystemen führt über eine präzisierte operationelle Semantik zu einer Begründung bzw. Verbesserung von auf der induktiven Zusicherungsmethode basierenden Verifikationsmethoden. Neben der korrekten Spezifikation der Bedeutung von Programm-Moduln umfaßt dabei der Begriff der partiellen Korrektheit auch das Nichtabblocken von Berechnungen. Der Nachweis der Terminierung eines Schleifen-Moduls wird darauf zurückgeführt, daß sich eine jeweils gegebene Schleifeninvariante in zwei geeignete intermittierende Zusicherungen zerlegen läßt.

INHALTSVERZEICHNIS Seite

1. Einleitung 1

2. Anwenderbereiche 14

3. Eine auf Anwenderbereichen aufbauende Modellbildung von Rechensystemen 31

3.1 Algorithmische Modellbildung 31

3.1.1 Pragmatik, Syntax und Semantik "von-Neumann-ähnlicher" Sprachen 31

3.1.1.1 Pragmatische Vorüberlegungen zur Klassifizierung und Strukturierung von Programm-Bausteinen 31

3.1.1.2 Syntax 40

3.1.1.3 Graphentheoretische Modellbildung 84

3.1.1.4 Semantik 92

3.1.2 Verifikation von Programm-Systemen 100

3.1.2.1 Partielle Korrektheit 104

3.1.2.1.1 Theoretische Grundlagen 104

3.1.2.1.2 Verifikationskalküle und -Methoden 110

3.1.2.2 Terminierung 137

3.2 Architektonische Modellbildung 145

4. Stichwortverzeichnis 157

5. Literaturverzeichnis 159

1. Einleitung

> *"The engineer is always embedded in his environment and works within a particular tradition. Sometimes the very intensity with which a problem is attacked conceals a simple solution, and only a fresh look at the problem leads to progress."*
>
> Wolfgang Händler [1977]

Einerseits eröffnet die fortschreitende Entwicklung billigerer und kleinerer standardisierter LSI-Bausteine neue technologische Dimensionen für die Arbeitsweise und Strukturierung von Computern (z. B. Händler [1974] , Händler/Hofmann/Schneider [1974]), andererseits beklagt u. a. Bauer [1978] den Zwang zur "Atomisierung und Sequenzialisierung" von Algorithmen beim Einsatz herkömmlicher Computer und Gagliardi [1974], Boulaye [1975] und Dennis [1976] werfen die Frage auf, ob nicht implizit als unveränderlich bzw. irrelevant vorausgesetzte Rechnerarchitekturen bei der Diskussion drängender Software-Probleme den Blick für potentiell mögliche Lösungsansätze versperren. Daneben gibt es eine wachsende Zahl an Vorschlägen für Spezialrechner bzw. "Software-orientierte Rechnerarchitekturen" (z. B. Grimsdale [1976], Berkling [1978], Bürkle/Frick/Schlier [1978]) und nicht nur Hirschberg [1977] spricht vom Ende einer Strukturepoche bei der Rechneranwendung und sieht einen Übergang zur Verwendung von Spezialrechnern und Multiprozessoren und zu einer Integration von Hardware und Software.

Es gibt also genug Gründe, die herkömmliche Arbeitsweise und Strukturierung von Systemen zur automatischen Bearbeitung von Rechenaufgaben (kurz Rechensystemen) zumindest theoretisch

einer Revision zu unterziehen und sich im folgenden einer Theorie von Rechensystemen zuzuwenden, bei der die herkömmliche Strukturierung von Rechensystemen in aufgabenunspezifische Universalrechner und aufgabenspezifische Anwender-Software kein Dogma mehr ist, bei der die Vorteile einer klaren Orientierung des Rechensystems an den jeweiligen Anforderungen der Anwender nutzbar werden, bei der nicht die technologie-abhängige Schnittstelle zwischen Hard- und Software, sondern vielmehr die weitgehend technologie-invariante Schnittstelle zwischen anwenderinvarianten und anwenderspezifischen Komponenten von Rechensystemen Kristallisationspunkt ist und bei der die Frage nach technologischen Realisierungsmöglichkeiten zunächst weitestgehend ausgeklammert wird.
Da der "Spezialfall" der herkömmlichen technologischen Realisierung von Rechensystemen mittels Universalrechnern in diese Theoriebildung partiell einbettbar sein soll, will die vorliegende Arbeit auch einen Beitrag dazu leisten, einige als "Fragen des Kunstgeschmacks" verstandene Aspekte der Informatik einer mehr wissenschaftlichen Betrachtungsweise zuzuführen.

Die Beschränkung der Theoriebildung auf Rechensysteme geht in Anlehnung an die traditionelle Unterscheidung zwischen naturwissenschaftlich-technischen, kaufmännischen und realtime-orientierten Aufgaben von einer Differenzierung zwischen Rechen-, Verwaltungs- und Steuerungsaufgaben aus. Das Charakteristikum von Rechenaufgaben wird hierbei in (langen) Ketten auszuführender Rechenoperationen ("Berechnungen") gesehen, das von Verwaltungsaufgaben in der Verwaltung vieler und evtl. komplex strukturierter Daten und das von Steuerungsaufgaben in komplexen zeitlichen Koordinierungsbedingungen. Natürlich ist insbesondere der Übergang zwischen Rechen- und Verwaltungsaufgaben fließend, aber derartige fließende Übergänge gibt es bei vielen wissenschaftlichen Begriffsbildungen, die sich als sinnvoll bzw. nützlich erwiesen haben.
Die Zweckmäßigkeit einer Differenzierung zwischen Rechen-, Verwaltungs- und Steuerungssystemen resultiert hauptsächlich aus theorie-orientierten Überlegungen: Zum einen erschien es entsprechend dem Prinzip "teile und herrsche" erfolgversprechender

anstelle eines Beitrags zu einer globalen ingenieur-wissenschaftlichen Theorie nur einen Beitrag zur Theorie von Rechensystemen zu leisten, zum anderen legten existierende Modellbildungen (Programm-Schemata, Datenbank-Modelle und Petri-Netze) sowie eigene theoretische Vorarbeiten auf dem Gebiet der Programm-Verifikation (Luft [1976]) diese Differenzierung nahe. Da die Strukturierung von zu automatisierenden Aufgaben in Rechen-, Verwaltungs- und Steuerungs-Moduln Entwurf, Implementierung, Verifikation und Wartung des resultierenden Systems erleichtern dürfte, wird mit unserer Differenzierung zwischen Rechen-, Verwaltungs- und Steuerungssystemen jedoch auch direkt ein praktischer Nutzen verknüpfbar sein.

Obwohl mit der Beschränkung auf Rechensysteme bereits eine Vielzahl von Fragestellungen ausgeklammert wird, ist die Klasse aller nur denkbaren Rechensysteme noch zu mächtig, um hier im Rahmen einer einheitlichen Theoriebildung systematisch behandelbar zu sein. So wird sich denn unsere Theoriebildung auch auf "von-Neumann-ähnliche" Rechensysteme[1)] beschränken, genauer auf Rechensysteme für die Bearbeitung nicht-rekursiver Algorithmen ("Flußdiagramm-Programme"). Die Bedeutung anderer denkbarer Rechensysteme wie vor allem "rekursiver" bzw. "applikativer" Rechensysteme, die z. B. von Bauer [1978] bzw.Backus [1978] favorisiert werden, soll damit jedoch nicht geschmälert werden: Es ist naheliegend, daß es das optimale Rechensystem für alle Aufgabenstellungen nicht geben wird, d. h. mehrere alternative Rechensysteme dürften ihre Berechtigung besitzen. Nicht zuletzt aufgrund der geschichtlichen Entwicklung und der investierten Summen in die Programm-Entwicklung dürften "von Neumann-ähnliche" Rechensysteme jedoch noch lange Zeit die wichtigsten Systeme zur Automatisierung von Rechenaufgaben bleiben.

---

1) Während einerseits in einer Vielzahl neuer Rechnerarchitekturen ein Übergang zur "Post-von Neumann-Ära" gesehen wird (z.B. Händler [1974], Bürkle/Frick/Schlier [1978]), sieht Backus [1978] das Charakteristikum der "von Neumann-Ära" in der Verwendung von Programmiersprachen mit Variablen, Kontrollanweisungen und Wertzuweisungen ("von Neumann-Sprachen"). Da unsere Rechensysteme einerseits auf "von Neumann-Sprachen" abzielen, u. a. jedoch mit der Verwendung von Mehrfach-Wertzuweisungen und parallelen Klauseln das Modell des von Neumann-Rechners sprengen, sprechen wir von "von Neumann-ähnlichen Rechensystemen".

Im Mittelpunkt einer ingenieurwissenschaftlichen Theorie von Rechensystemen steht der Prozeß der Synthese bzw. des Entwurfs von Rechensystemen, wobei analog zu anderen ingenieurwissenschaftlichen Entwurfs-Prozessen auch der Entwurf von Rechensystemen nicht als zweck- und damit wertfrei[1] angesehen werden kann. Es geht also auch beim Entwurf von Rechensystemen darum, ausgehend von (mehr oder weniger) präzisierten Anforderungen (Zwecken) von Anwendern bzw. Anwenderklassen unter Verwendung gegebener technologischer Bausteine bzw. über den Entwurf neuer technologischer Bausteine Mittel bereitzustellen, die bezüglich dem derzeitigen Stand der Technik den gestellten Zwecken "optimal" dienen.
Eine ingenieurwissenschaftliche Theorie von Rechensystemen baut somit an einer Brücke, die von Aufgaben-/Problem-Beschreibungen in Form anwenderorientierter Sprachen und Modelle zu möglichen bzw. optimalen physikalisch-technischen Aufgaben-/Problem-Lösungen führt.

Unser Beitrag zu diesem Brückenbau konzentriert sich dabei auf zwei Brückenpfeiler in Form einer algorithmischen und einer architektonischen Modellbildung, auf die sich einerseits das anwendungsspezifische System-, Programm- und Moduldesign[2] und andererseits auch die technologische Realisierung beziehen können.
Nicht nur die algorithmische, sondern auch die architektonische Modellbildung werden primär anwenderorientiert sein, wobei die Charakterisierung unserer Modelle als "anwenderorientiert" nicht eine Vertrautheit derzeitiger Rechensystem-Anwender mit unserer Modellbildung meint. Vielmehr sind unsere Modelle dadurch anwenderorientiert, daß sie neben einem feststehenden anwenderunabhängigen Bestandteil auch einen jeweils anwenderspezifischen Bestandteil besitzen, der sich der jeweiligen Sprache eines Anwenders (bzw. wissenschaftlichen Fachsprache) direkt anpassen läßt.

---

1) Zur Beziehung zwischen Werten, Zwecken und Mitteln siehe z. B. Lorenzen/Schwemmer [1973].

2) Die Unterscheidung zwischen System-, Programm- und Moduldesign orientiert sich an Sneed [1977].

Der Aufbau unserer Modelle lehnt sich damit an den Aufbau von Sprachen der formalen Logik (wissenschaftliche Fachsprachen)[1] an, wo klar zwischen anwendungsinvarianten Formen und anwendungsspezifischen Bedeutungen ("Inhalten") unterschieden wird[2]: Unserer algorithmischen Modellbildung liegt eine einheitliche anwendungsinvariante syntaktische und semantische Struktur zugrunde und da unsere architektonische Modellbildung primär einer realisierungsbezogenen Widerspiegelung der operationellen Semantik gegebener Programmsysteme dient[3], impliziert die anwendungsinvariante semantische Struktur der algorithmischen Modellbildung auch eine anwendungsinvariante Struktur unserer architektonischen Modelle. Der jeweilige "Inhalt" unserer algorithmischen Modellbildung wird in Form von "Anwenderbereichen" und der jeweilige "Inhalt" der architektonischen Modellbildung aus den diesen "Anwenderbereichen" zugeordneten speziellen (bzw. speziell programmierten) Prozessor/Speicher-Elementen bestehen. Neben einer realisierungsbezogenen Widerspiegelung der operationellen Sementik gegebener Programmsysteme dient unsere architektonische Modellbildung jedoch auch zur Darstellung anwenderspezifischer Durchsatzforderungen. Unsere Theoriebildung konzentriert sich also auf den Prozeß der Synthese anwenderorientierter Rechenprogramme[4] und den Prozeß der Synthese "passender" Rechnerarchitekturen, d. h. auf zwei ausgezeichnete Schnittstellen bei der Synthese von Rechensystemen.

---

1) Siehe z. B. Oberschelp [1974].

2) Eine pragmatisch orientierte Liberalisierung der Forderung nach einheitlicher anwenderinvarianter Modellbildung mittels der Verwendung anwendungsspezifischer Abkürzungen und Notationen ist jedoch auch hier analog zur Verwendung wissenschaftlicher Fachsprachen zweckmäßig.

3) Auf den Zusammenhang zwischen der operationellen Semantik von Programmiersprachen und der architektonischen Modellierung von Rechensystemen weist neben Tannenbaum [1978] auch Backus [1978] hin: "Underlying every programming language is a model of a computing system that its programs control." (pp. 614)

4) Eine Unterscheidung zwischen Rechenalgorithmus (= Programm für abstrakten Rechner) und Rechenprogramm (= Programm für realen, evtl. über Compiler definierten Rechner) ist im Rahmen unserer Theoriebildung hinfällig.

Mit der Unterscheidung zwischen ingenieurwissenschaftlichen Theorien, die sich einerseits mehr auf die Seite der Aufgaben-/ Problem-Beschreibung und andererseits mehr auf die Seite der physikalisch-technischen Realisierung beziehen, wird auch eine Arbeitsteilung zwischen Ingenieuren nahegelegt.
Angesichts von in "Mannjahren" geschätzten komplexen Automatisierungsaufgaben ist ein Zwang zur Arbeitsteilung ohnehin vorhanden und es ist naheliegend, diese Arbeitsteilung auch an den verschiedenen Verfeinerungs-Niveaus bei der Realisierung von Problem-Lösungen zu orientieren: Häufig ist auf den Stufen hoher Abstraktion der Stand der Technik nur in Form globaler Preis-/Leistungs-Parameter zu berücksichtigen, während auf den unteren Verfeinerungs-Niveaus in der Regel detaillierte Technologiekenntnisse erforderlich sind.
Die Frage nach einer Methodik für die Synthese von Rechensystemen orientiert sich u. a. auch an allgemeinen Methodiken des Problemlösens (siehe z. B. Polya [1973]), d. h. an Erkenntnissen aus physiologischen, psychologischen, pädagogischen und wissenschaftstheoretischen Fragestellungen zum Problemlösen. Viele "Software-Methoden" (siehe z. B. Weinberg [1971], Dijkstra [1972], Wirth [1972]) hätten sich zum Teil auch aus allgemeinen Erkenntnissen zum Problemlösen ableiten lassen!

Der Prozeß der Synthese eines Rechenprogramms läßt sich bei geeigneter Schnittstellennormierung weitgehend auf die Synthese einzelner Programm-Moduln reduzieren. Eine weitergehende Aufgliederung der Synthese von Programm-Moduln in Entwurf, Implementierung und Verifikation kann wegen bestehender Zusammenhänge und Rückkopplungen keine zeitliche Abfolge implizieren. Abstrahieren wir von den Interdependenzen, so bezeichnet der Entwurf den Prozeß der präzisen Aufgaben-/Problem-Beschreibung und der Erstellung eines Lösungs-

Plans mit einer Spezifizierung struktureller und funktioneller Eigenschaften, die Implementierung den Prozeß der Verknüpfung von Bausteinen zur (evtl. abstrakten) Realisierung des Entwurfs und die Verifikation den Prozeß, der den Nachweis liefert, daß die Implementierung die im Entwurf spezifizierten Eigenschaften erfüllt.

Die Interdependenzen zwischen Entwurf, Implementierung und Verifikation wollen wir u. a. dadurch berücksichtigen, daß ähnlich wie in anderen "Programmierlogiken" (z. B. Constable [1977]) unsere Programme sowohl aus Anweisungen zur Veränderung von Programmzuständen, als auch aus Zusicherungen (assertions) zur Beschreibung von Programmzuständen bestehen werden und darüber hinaus der Entwurf bzw. das Verständnis der mit den Programmen (evtl. nur implizit) verknüpften Beweise durch einen strukturierten, modularen Programm-Aufbau erleichtert wird. Die Zweckmäßigkeit einer Erweiterung des Programmbegriffs um Zusicherungen ergab sich dabei unabhängig von der Entwicklung der Programmiersprache Euclid (z.B. Lampson/Horning/u.a. [1977]) aus eigenen Vorarbeiten zur Methodik der Programmverifikation während der Programmerstellung (Luft [1976]).

Bei der Spezifikation unserer Rechenalgorithmen in Form von Zusicherungen wollen wir uns - vereinfacht dargestellt - auf Sprachen der Quantoren-Logik (first-order-logic) beschränken, obwohl es sich hierbei um keine "vollständigen" Spezifikationssprachen handelt, d. h. die Formalisierung aller wünschenswerten Zusicherungen (Schleifeninvarianten) mittels dieser Sprachen nicht möglich ist (siehe z. B. Wand [1976]). Auf "präalgorithmische Formulierungen mittels Auswahl und Kennzeichnung" (Bauer [1977] S. 83) wird also ebenso verzichtet wie auf prädikative Formeln ($X \in Y$) und auf Abstraktionsterme ($\{x \mid \Phi(x)\}$[1]). Der Einsatz von Spezifikationssprachen, die über die üblichen Sprachen der Quantorenlogik hinausgehen bzw. die Verwendung von prädikativen Formeln, Auswahl-, Kennzeichnungs-, Paar- und Abstraktionstermen bei Rechenanweisungen

---

1) Siehe z. B. Oberschelp [1974] S. 29.

und deren Implementierungsmöglichkeiten (z. B. mithilfe assoziativer Speicher) erfordert eigenständige, tiefergehende Untersuchungen, die evtl. auch besser im Rahmen einer "Theorie von Verwaltungssystemen" zu erbringen sind.

Die Beschränkung auf Rechensysteme unter Ausklammerung von Steuerungsaufgaben führte nicht zu einem vollständigen Verzicht auf Möglichkeiten zur Parallelverarbeitung von Anweisungen. Die Struktur der in unserer Theoriebildung eingeschlossenen Parallelität ist jedoch so einfach, daß keine komplizierten zeitlichen Koordinierungsbedingungen auftreten können, die (abstrakte) Konfliktfreiheit rein syntaktisch überprüfbar ist und bezüglich der Forderung nach Korrektheit, Wartbarkeit und Änderungsfreundlichkeit keine Probleme auftreten.

Bevor auf den Inhalt der einzelnen Kapitel dieser Arbeit näher eingegangen wird, wollen wir noch kurz den Bezug zwischen unserer Theoriebildung und anderen Theorien aufzeigen. Unsere Theoriebildung kristallisierte sich anhand algorithmischer Sprachen, die direkt auf dem Konzept der formalen Logik zur Präzisierung wissenschaftlicher Fachsprachen aufbauen und klar zwischen anwendungsinvarianten und anwendungsspezifischen Strukturen unterscheiden (Luft [1976]). Eine derartige Anlehnung an die formale Logik und klare Unterscheidung zwischen anwendungsinvarianten und anwendungsspezifischen Strukturen ist nicht nur unter wissenschaftsimmanenten, sondern auch unter ingenieurwissenschaftlichen Gesichtspunkten vorteilhaft:

- Formale Verifikationsmethoden für die Korrektheit von Rechensystemen bauen auf Sprachen und Kalkülen der formalen Logik auf.
- Viele Fragen des Entwurfs, der Verifikation und Implementierung von Rechensystemen sind unabhängig von den konkreten Aufgaben, die das Rechensystem bearbeiten soll. Eine interaktive Rechnerunterstützung ("Programmierlaboratorium", "programmers work bench") für den Prozeß des Entwurfs, der

Verifikation und Implementierung von Rechensystemen sollte z. B. diesen Sachverhalt berücksichtigen und die Vorteile einer klaren Schnittstelle zwischen anwendungsinvarianten und anwendungsspezifischen Fragestellungen nutzen (siehe Luft [1978]).

Der Zusammenhang zwischen Sprachen der formalen Logik (wissenschaftlichen Fachsprachen) und unserer Theoriebildung ist dabei dadurch gegeben, daß wir algorithmische Sprachen als um "algorithmische Konstanten" (Wertzuweisung, Kontrollanweisungen) erweiterte "Sprachen der formalen Logik" auffassen.

Unter Hinzunahme ingenieurwissenschaftlicher Fragestellungen baut diese Betrachtungsweise auf den Arbeiten von Janov [1960] und Luckham/Park/Paterson [1970] auf, die mit dem Begriff "Programmschema" einen Weg zur Formalisierung von Algorithmen analog zur Formalisierung wissenschaftlicher Fachsprachen in speziellen Sprachen der formalen Logik aufgezeigt haben. Ähnlich wie in Sprachen der formalen Logik für Terme und Formeln ein einheitliches anwenderinvariantes Sprachgerüst bereitgestellt wird, so besitzen auch unsere algorithmischen Sprachen zur Formulierung anwenderorientierter Eigenschaften von Rechensystemen ein Sprachgerüst mit einheitlicher anwenderinvarianter Struktur, in das die jeweiligen anwenderspezifischen Bezeichnungen und Begriffe in normierter Weise eingebracht werden.

Im Unterschied zur Syntax herkömmlicher Programmiersprachen soll die Syntax-Definition unserer algorithmischen Sprachen in erster Linie ein präzises Fundament für nachfolgende theoretische Untersuchungen liefern und möglichst jeweils "auf einen Blick" über die syntaktische Struktur und den Aufbau

sprachlicher Bausteine informieren. Daraus ergibt sich zunächst einmal, daß wir zur Syntax-Definition eine konkrete Syntax verwenden sollten, zumal sich mit dem oben erwähnten Bezug auf ein anwenderinvariantes Sprachgerüst unsere Syntax-Definition bei Verwendung einer abstrakten Syntax nicht wesentlich vereinfachen würde.
Aus dem Zweck unserer Syntax-Definition folgt weiter, daß die Frage nach der Anwendbarkeit schneller Syntaxanalyse-Verfahren als zweitrangig angesehen wird und hierfür evtl. notwendige Einschränkungen bzw. Erweiterungen der Syntax-Definition erst in einem zweiten Schritt - nach der Präzisierung der wünschenswerten Eigenschaften - diskutiert und vollzogen werden sollten.
So wird denn auch zur Darstellung der syntaktischen Struktur unserer algorithmischen Sprachen das in der Mathematik übliche Definitionsschema verwendet. Obwohl die dadurch definierten sprachlichen Gebilde (Moduln, Rechenprogramme) offensichtlich nicht mittels kontextfreier Grammatiken erzeugbar sind, d. h. die üblichen Techniken der Syntaxanalyse nicht direkt anwendbar sind (siehe z. B. Aho/Ullman [1972]), scheint die automatische Überprüfung der implizit verwendeten kontextsensitiven Bedingungen und damit die Syntaxanalyse insgesamt leicht durchführbar zu sein. Auf die Ausarbeitung und Diskussion möglicher und guter Syntaxanalyse-Verfahren für unsere Rechenprogramme mußte jedoch verzichtet werden.

Die Semantik unserer algorithmischen Sprachen stellt in Form einer auf den semantischen Grundbegriffen der Logik (d. h. einer denotationellen Semantik für Terme und Formeln) aufbauenden operationellen Semantik eine "Standardsemantik" bzw. ein "semantisches Fundament" bereit, aus dem andere "Semantik-Ebenen", wie z. B. eine denotationelle Semantik oder die Hoare/Dijkstra'sche Prädikat-Transformation aufgebaut bzw. abgeleitet werden können. Man kann also die für unterschiedliche Zwecke jeweils geeigneten bzw. wünschenswerten Semantiken aus unserer operationellen Semantik ableiten. Der Bezug auf eine operationelle Semantik im Rahmen unserer Theoriebildung impliziert damit nicht, daß Programmierer auch diese unterste

"semantische Verfeinerungs-Ebene" kennenlernen müssen oder gar gezwungen sind "in Form mechanischer Details anstelle von Konzepten zu denken" (Milne/Strachey [1976] pp. 13).
Auf eine systematische Einarbeitung algebraischer Spezifikationsmethoden zur Datenabstraktion (Liskov/Zilles [1975]) bzw. abstrakter Datentypen (Guttag/Horowitz/Musser [1978]) in unsere Theoriebildung mußten wir verzichten. Mit dem Begriff des Anwenderbereichs, der weitgehend dem Begriff des Datentyps[1] entspricht, ist jedoch ein direkter Ansatzpunkt für eine Berücksichtigung der Theorie abstrakter Datentypen gegeben: Analog zur algebraischen Spezifikation von Datentypen könnten wir auch Anwenderbereiche algebraisch spezifizieren.
Die Relevanz unserer operationellen Semantik bzw. der Verifikationsmethoden nach der induktiven Zusicherungsmethode würde dadurch nicht beseitigt werden.

Die Orientierung unserer algorithmischen Sprachen an ingenieurwissenschaftlichen Fragestellungen wie z. B. nach der Handlichkeit, Zuverlässigkeit, Wartbarkeit und Änderungsfreundlichkeit sprachlicher Konstrukte (Anweisungen, Moduln, Programme) führte zu einer Berührung mit der Diskussion über strukturierte und modulare Programmierung und den hierfür geeigneten Sprachkonstrukten (siehe z. B. Dijkstra [1972], Wirth [1972], Blazert [1977], Schneider [1978]).

Aus dem obenstehenden geht hervor, daß das Schwergewicht unserer Theoriebildung auf den Prozeß der Synthese von Rechensystemen abzielt. Abgesehen von der Frage nach der Korrektheit werden Fragen zur Analyse anwenderorientierter Eigenschaften von Rechensystemen nur am Rande behandelt, wobei jedoch Zusammenhänge zwischen unserer Theoriebildung und einer mehr analytisch orientierten, wie z. B. der von Halstead [1977], relativ leicht aufzeigbar sind.

---

1) In Anlehnung an Guttag/Horowitz/Musser [1978] verstehen wir unter einem Datentyp eine Menge von Werten zusammen mit einer Menge von Operationen auf diesen Werten.

Bei einer Arbeit, die sich nicht nur in einen vorgegebenen begrifflichen Rahmen einordnen, sondern diesen begrifflichen Rahmen präzisieren und verbessern will, bereitet Darstellung und Verständnis der Arbeit besondere Schwierigkeiten.
Neben unpräzisiert verwendeten wissenschaftlichen Fachbegriffen (LSI-Bausteine, Algorithmus, Rechnerarchitektur, Programm-Modul, Berechnung u. a.) gibt es bereits in der Einleitung eine zweite sprachliche Ebene umgangssprachlich präzisierter bzw. neu eingeführter wissenschaftlicher Fachbegriffe, die den neuen begrifflichen Rahmen der Arbeit grob abstecken (Rechensystem, Rechen-, Verwaltungs- und Steuerungsaufgaben, von-Neumann-ähnliche Rechensysteme, ingenieurwissenschaftliche Theorie, anwenderorientiert u. a.). Ein Teil der zunächst unpräzisiert verwendeten wissenschaftlichen Fachbegriffe wie z. B. Programm-Modul, Programm-System und Berechnung wird später beim Aufbau einer präzisen algorithmischen Modellbildung, d. h. bei der Definition einer Objektsprache, rekonstruiert werden. Da diese Objektsprache u. a. aus sprachlichen Objekten (Terme, Moduln, Programm-Systeme) besteht, die nicht nur gebraucht sondern auch gemeint werden, wird die Darstellung und das Verständnis der Arbeit zusätzlich kompliziert. Außerdem werden zum Aufbau dieser Objektsprache präzis definierte metasprachliche Begriffsbildungen wie z. B. die Wertstelligkeit wst(t) eines Terms t verwendet.
Zur Unterscheidung zwischen dem Gebrauchen und dem Meinen objektsprachlicher Ausdrücke (z. B. Terme und Moduln) werden gelegentlich Anführungszeichen bzw. Quasianführungszeichen (siehe z. B. Oberschelp [1974]) verwendet. Da aus dem Kontext in der Regel direkt ersichtlich ist, ob Zeichenreihen gemeint oder gebraucht werden, wird auf eine systematische Verwendung der Anführungszeichen jedoch verzichtet.

Die Beschränkung des Umfangs dieser Arbeit auf ein akzeptierbares Maß führte insbesondere auch zu einem Verzicht auf eine exemplarische Erläuterung von Verifikationsmethoden anhand ausgewählter Programm-Beispiele. Die präzisierten und verbesserten Verifi-

kationsmethoden nach der induktiven Zusicherungsmethode kann ein mit der Programmverifikation und den Grundbegriffen der formalen Logik vertrauter Fachmann jedoch auch ohne Programm-Beispiele verstehen.
Eine realitätsbezogene Überprüfung (und damit einhergehende Verbesserung) unserer Theoriebildung können "übliche" kleine Programm-Beispiele nur sehr begrenzt leisten.

Mein besonderer Dank gebührt den Herren Prof. Dr. W. Händler, Prof. Dr. H. Müller und Prof. Dr. H.J. Schneider, die mir die Durchführung dieser Arbeit ermöglichten und zahlreiche nützliche Anregungen gaben. Danken möchte ich insbesondere auch den Herren Dr. H. Hessenauer und Dr. M. Nagl für ihre Diskussionsbereitschaft sowie meiner Ehefrau Helene für die Reinschrift des Manuskriptes.

2. Anwenderbereiche

Wie schon in der Einleitung erwähnt wurde, wollen wir unsere algorithmischen Sprachen auf Sprachen der formalen Logik bzw. den dort benutzten Begriffsbildungen aufbauen. Ein wesentlicher Unterschied zu den Sprachen der mehrsortigen klassischen Logik bzw. zu Oberschelp [1974] ist dabei zunächst der Verzicht auf die Auszeichnung von Standardobjekten (z. B. Wahrheitswerte, natürliche Zahlen). Die Auszeichnung von Standardobjekten sowie zugehörigen Standardkonstanten und Standardfunktionen würde im Rahmen unserer Theoriebildung bedeuten, daß jedes abstrakte Maschinenmodell einer Programmiersprache Standard-Speicherelemente und Standard-Operationen besitzen müßte.
Nicht nur im Hinblick auf Anwendungsgebiete für Mikroprozessoren gibt es jedoch relevante abstrakte Maschinenmodelle, die z. B. über keine natürlichen oder rationalen Zahlen verfügen.
Eine standardmäßige Auszeichnung von z. B. natürlichen Zahlen und arithmetischen Operationen bereitet in unserer Theorie jedoch keine Schwierigkeiten und führt zu ausgezeichneten Teilklassen der hier betrachteten algorithmischen Sprachen.

Ein zweiter wesentlicher Unterschied zu Sprachen der klassischen Logik ist der Verzicht auf Relationskonstanten, der zum einen verknüpft ist mit dem Verzicht auf Standardbereiche, da Rel.kon. den Standardbereich der Wahrheitswerte voraussetzen. Außerdem sind aus der operationellen Sicht eines Rechners Relationen spezielle Funktionen, die Wahrheitswerte liefern, d. h. auch im Hinblick auf die Implementierung ist eine Auszeichnung von Relationen nicht zu rechtfertigen. Nicht zuletzt vereinfacht sich durch den Verzicht auf Relationskonstanten, d. h. mit der Beschränkung auf "funktionale" Sprachen, der Aufbau und die Darstellung unserer Theoriebildung.
Die durch den Verzicht auf Relationskonstanten verminderte Anwendernähe kann dabei gegebenenfalls mit der Einführung von Abkürzungen und Standardinterpretationen wieder hergestellt werden.

Ein dritter wesentlicher Unterschied zu Sprachen der klassischen Logik ist die besondere Rolle von Termen, die wir hier nicht nur zum Aufbau von Formeln, sondern auch zur Darstellung von Wertzuweisungen und Abfragen benötigen.

Ein vierter wesentlicher Unterschied ist die Verwendung partieller Interpretationen mit dem Wert "undefiniert" (Ω). Dabei werden wir sowohl bei der Definition des Wertes von Termen, als auch bei der Definition des Wertes von Formeln bezüglich dem Wert Ω eine "Standarderweiterung" verwenden, die stets den Wert Ω liefert, wenn ein Teilausdruck undefiniert ist. Andere Erweiterungsmöglichkeiten werden z. B. in Löffler [1977] diskutiert.

Ein fünfter wesentlicher Unterschied besteht in der Verwendung zweier verschiedener Bedeutungs-Ebenen für die logischen Operatoren[1]: Neben der Darstellung üblicher logischer Zusammenhänge in Form von Zusicherungen können logische Operatoren auch bei Abfragen oder Wertzuweisungen zur Berechnung von Wahrheitswerten verwendet werden.

Ein sechster wesentlicher Unterschied besteht in der Verwendung von (m,n)-stelligen anstelle von (m,1)-stelligen Funktionskonstanten: Neben der Möglichkeit zur kompakteren Programmdarstellung sind (m,n)-stellige Funktionskonstanten mit $n>1$ auch deshalb vorteilhaft, weil viele Rechenwerks-Befehle herkömmlicher Rechenmaschinen mehrstellige Ergebnisse liefern[2] und diese (auch als unerwünschte"Seiten-Effekte"!) im Programmtext direkt sichtbar sein sollten. Nicht zuletzt sind beim derzeitigen Stand der Technik viele Funktionskonstanten als Funktionsprozeduren gegebener Programmiersprachen implementiert und eine Beschränkung auf (m,1)-stellige Funktionskonstanten würde mit einer Beschränkung auf (m,1)-stellige Funktionsprozeduren und den damit verbundenen Nachteilen einhergehen.

---

1) $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.

2) Z. B. bei der ganzzahligen Division.

Def. 1

Ein Anwenderalphabet A besteht aus der Vereinigung folgender Zeichenmengen:

i) Der Menge der Bereichskonstanten BKonst(A) von A.

ii) Der Menge der (m,n)-stelligen Funktionskonstanten $\overset{(m,n)}{\text{FKonst}}(A)$ von A mit $m,n \in N_o \times N$,

$$(\forall\, m_1,m_2 \in N_o)(\forall\, n_1,n_2 \in N)(m_1 \neq m_2 \vee n_1 \neq n_2 \rightarrow \overset{(m_1,n_1)}{\text{FKonst}}(A) \cap \overset{(m_2,n_2)}{\text{FKonst}}(A) = \emptyset)$$

und

$$(\forall\, n \in N)\ \text{Var}(A) \cap \overset{(o,n)}{\text{FKonst}}(A) = \emptyset.$$

iii) Der Menge der Individuenvariablen Var(A) von A.

Bezeichnung

(O,n)-stellige Funktionskonstanten nennen wir auch Individuenkonstanten. Eine Konstante von A ist eine Bereichs- oder Funktionskonstante von A, eine Variable von A eine Individuenvariable von A.

Beispiel

Fragen wir danach, ob evtl. ALGOL 60 implizit ein Anwenderalphabet zugrunde liegt, so stellen wir fest, daß die Forderung nach Disjunktheit der Funktionskonstanten dort anscheinend nicht erfüllt ist, da "+" und "-" in Abhängigkeit vom Kontext einmal als Vorzeichen einer Zahl (einstellige Funktionskonstante), ein andermal als Additions- bzw. Subtraktionsfunktion (zweistellige Funktionskonstante) verwendbar ist. Berücksichtigen wir jedoch, daß die Verwendung von "+" als Vorzeichen überflüssig ist (Identitätsfunktion) und fassen wir "-(⌜Term⌝)" als Abkürzung für "(O-⌜Term⌝)" auf, so erhalten wir als Anwenderalphabet A von ALGOL 60:

$$\mathrm{BKonst}(A) = \{\underline{\mathrm{boolean}}, \underline{\mathrm{integer}}, \underline{\mathrm{real}}\}$$

$$\mathrm{Var}(A) \cong \{x \mid x \text{ ist ALGOL 60-Variable}\}^{1)}$$

$$\mathrm{FKonst}^{(0,1)}(A) = \{\underline{\mathrm{true}}, \underline{\mathrm{false}}\} \cup \{x \mid x \text{ ist ALGOL 60-Zahl}\}$$

$$\mathrm{FKonst}^{(1,1)}(A) = \{\underline{\mathrm{not}}\}$$

$$\mathrm{FKonst}^{(2,1)}(A) = \{+,-,\times,/,\dot{-},\uparrow,$$
$$<,\leqq,>,\geqq,=,$$
$$\underline{\mathrm{and}}, \underline{\mathrm{or}}, \underline{\mathrm{impl}}, \underline{\mathrm{equiv}}\}$$

$$\mathrm{FKonst}^{(m,n)}(A) = \emptyset \text{ für } m>2 \vee n>1.$$

Bemerkung

Während eine Unterscheidung zwischen Bereichskonstanten und Individuenvariablen durch den völlig verschiedenen Kontext ihrer Verwendung innerhalb von Programmen "auf einen Blick" möglich ist, ist eine kontextabhängige Unterscheidung zwischen argumentfreien (0,n)-stelligen Funktionskonstanten und Individuenvariablen nicht immer[2)] und eine kontextabhängige Unterscheidung zwischen (m_1,n)- und (m_2,n)-stelligen Funktionskonstanten bzw. zwischen (m,n_1)- und (m,n_2)-stelligen Funktionskonstanten evtl. nur unter Abzählen von Termlisten und unter Reduzierung der Sicherheit gegenüber Programmierfehlern möglich. Die Forderung nach disjunkten Namen für Funktionskonstanten - auch bei unterschiedlicher Stellenzahl - erleichtert somit die Programmlesbarkeit und ermöglicht das Erkennen einer speziellen Klasse von Schreibfehlern bei der Syntaxanalyse.

---

1) Zwischen unserem Variablenbegriff und ALGOL 60-Variablen besteht lediglich eine Ähnlichkeit (z. B. gibt es bei uns keine indizierten Variablen: unsere Variablen können auch Felder als Wertebereiche besitzen, wobei der Zugriff auf Feldkomponenten mithilfe von Funktionen erfolgt).

2) In Termen kann bei gleicher Namenswahl zwischen (0,1)-stelligen Funktionskonstanten und Individuenvariablen nicht unterschieden werden.

Def. 2

A sei ein beliebiges Anwenderalphabet.

a) Eine Konstanten-Interpretation J_K von A besteht aus den folgenden Zuordnungen:

i) Jedem $b \in BKonst(A)$ wird eine nichtleere, endliche Objektmenge $J_K(b)$ zugeordnet.

ii) Jeder (0,n)-stelligen Funktionskonstanten f wird ein n-Tupel $(b_1(f), b_2(f), \ldots, b_n(f))$ von Bereichskonstanten aus BKonst(A) und ein Element d aus $J_K(b_1(f)) \times J_K(b_2(f)) \times \ldots \times J_K(b_n(f))$ zugeordnet. $(b_1(f), \ldots, b_n(f))$ heißt Bereichskennzeichnung von f bezüglich J_K (kurz $b(f, J_K)$) und das Element d heißt Wert von f bezüglich J_K (kurz $w(f, J_K)$).

iii) Jeder (m,n)-stelligen Funktionskonstanten f mit $m \geq 1$ wird ein (m+n)-Tupel $(b_1(f), b_2(f), \ldots, b_m(f), b_{m+1}(f), \ldots, b_{m+n}(f))$ von Bereichskonstanten aus BKonst(A) und eine berechenbare Funktion

$$g : J_K(b_1(f)) \times \ldots \times J_K(b_m(f)) \to J_K(b_{m+1}(f)) \times \ldots \times J_K(b_{m+n}(f)) \cup \{\Omega\}$$

zugeordnet.
$(b_1(f), \ldots, b_{m+n}(f))$ heißt Bereichskennzeichnung von f bezüglich J_K (kurz $b(f, J_K)$) und die Funktion g heißt Wert von f bezüglich J_K (kurz $w(f, J_K)$).

b) Eine Variablen-Interpretation J_V von A ordnet jeder Individuenvariablen $v \in Var(A)$ ein Element b aus BKonst(A) zu. b heißt Bereichskennzeichnung von v bezüglich J_V (kurz $b(v, J_V)$).

c) Eine <u>vollständige Interpretation eines Anwenderalphabets A</u> ist ein Paar (J_K, J_V), wobei J_K eine Konstanten-Interpretation von A und J_V eine Variablen-Interpretation von A ist.

<u>Bemerkung</u>

Die Unterscheidung zwischen der Konstanten- und der Variablen-Interpretation eines Anwenderalphabets orientiert sich bereits am Prozeß der Programmierung: Bei gleichem zugrundeliegenden Anwenderalphabet und gleicher Konstanten-Interpretation sollten bei Programm-Moduln unterschiedliche Variablen-Interpretationen möglich sein. Dies ist z. B. bei ALGOL mittels Typ-Vereinbarungen der Fall, während bei FORTRAN allen Variablen statisch (in Abhängigkeit vom Anfangsbuchstaben) ein Typ (eine Bereichskonstante <u>integer</u> oder <u>real</u>) zugeordnet ist. Sowohl die Konstanten- als auch die Variablen-Interpretation der Anwenderalphabete eines Moduls wird einheitlich an einer normierten Stelle im Modul (Modulkopf) erfolgen. Die Frage nach der Zweckmäßigkeit von Interpretationen, die sich nicht einheitlich auf alle Teile eines vollständig spezifizierten Moduls beziehen, sondern lokal für einzelne ausgezeichnete Teile jeweils speziell eingeführt werden (wie z. B. innerhalb einer Zusicherung mittels $(\forall\ v \in b)\Phi$), wird ausgeklammert.

<u>Def. 3</u>

Ein <u>Anwenderbereich</u> ist ein Paar (A,J), wobei A ein Anwenderalphabet und J eine Konstanten-Interpretation von A ist.

<u>Bemerkung</u>

Die Realisierung von Anwenderbereichen kann sowohl problemorientiert unter Rückgriff auf durch die Problemstellung vorgegebene "feinere" Anwenderbereiche als auch technologieorientiert erfolgen. Während sich die derzeitige technologie-

orientierte Realisierung von Anwenderbereichen weitgehend auf bereits zur Verfügung stehende "Maschinenbereiche" in Form von "Anwenderbereichen" gegebener Programmiersprachen, Mikroprogrammen und Standard-Software abstützt, zeigt die Entwicklung von Spezialrechnern einen Übergang zu einer gezielteren Beteiligung von Hardware an der Realisierung von Anwenderbereichen. Ein Ansatz, der die Nachteile von Spezialrechnern vermeiden will ohne die Vorteile einer gezielten Beteiligung von Hardware an der Realisierung von Anwenderbereichen aufgeben zu müssen, kann in dem Konzept konfigurierbarer Rechner (siehe z. B. Miller/Locke [1975]) gesehen werden.

Def. 4

A,A' seien Anwenderalphabete und J,J' Konstanten-Interpretationen von A bzw. A'.

a) A ist enthalten in A' (kurz $A \subseteq A'$) $\leftrightarrow$

$$BKonst(A) \subseteq BKonst(A') \wedge Var(A) \subseteq Var(A') \wedge$$
$$(\forall\, m,n \in N_0)\, FKonst^{(m,n)}(A) \subseteq FKonst^{(m,n)}(A').$$

b) (A,J) ist enthalten in (A',J') (kurz $(A,J) \subseteq (A',J')$) $\leftrightarrow$

$$A \subseteq A' \wedge (\forall\, b \in BKonst(A))\, J(b) \subseteq J'(b) \wedge$$
$$(\forall\, f \in FKonst^{(m,n)}(A))((m = 0 \rightarrow b(f,J) = b(f,J') \wedge w(f,J) = w(f,J'))$$
$$\wedge (m > 0 \rightarrow b(f,J) = b(f,J') \wedge w(f,J) \subseteq w(f,J'))).$$

Bemerkung

Unsere Definition der Enthaltensein-Relation von Anwenderbereichen bezieht sich nicht auf Fragen der Programm-Kompatibilität, sondern auf den Prozeß der Programm-Verifikation, bei dem neben einem "Maschinenbereich" (A,J) jeweils ein "Verifikations-Bereich" (A',J') mit $(A,J) \subseteq (A',J')$ betrachtet wird.

Def. 5

A sei ein Anwenderalphabet und $J = (J_K, J_V)$ eine vollständige Interpretation von A.
Eine Belegung σ über J (kurz $\sigma \in \Sigma_J$) ordnet jedem $v \in Var(A)$ ein Element $\sigma(v) \in J_K(b(v,J_V)) \cup \{\Omega\}$ zu.
Σ_J sei die Menge aller Belegungen σ über J.

Bemerkung

$b(v,J_V)$ liefert eine Bereichskonstante $b \in BKonst(A)$ und deren anschließende Interpretation $J_K(b)$ einen dieser Bereichskonstanten zugeordneten Grundbereich. In der Regel benutzt man beschränkte Belegungen, bei denen nur einigen Variablen aus Var(A) ein von Ω verschiedenes Element eines Grundbereichs zugeordnet ist. Der Wert der übrigen Individuenvariablen sei dann stets undefiniert (Ω).

Def. 6

$HZ \leftrightharpoons \{"(", ")", ","\}$ ist eine Menge von Hilfszeichen.

Bemerkung

Zum Aufbau von Termen benötigen wir noch Klammern und ein Trennzeichen.
Die Zeichen "(", ")" und "," haben in unserer Objektsprache eine Standardinterpretation als Klammern und Trennzeichen und dürfen deshalb, zur Vermeidung von Mehrdeutigkeiten, nicht in Anwenderalphabeten benutzt werden.
Sei also im folgenden A ein beliebiges Anwenderalphabet mit $A \cap HZ = \emptyset$ und $J = (J_K, J_V)$ eine vollständige Interpretation von A.

Def. 7

Die Menge der Terme über A (kurz Term(A)), sowie die Variablenmenge Var(t), die Funktionskonstantenmenge FKonst(t) und die Wertstelligkeit wst(t) eines Terms $t \in \mathrm{Term}(A)$ sind induktiv definiert:

i) α) Für $x \in \mathrm{Var}(A)$ gilt:

$x \in \mathrm{Term}(A)$,

$\mathrm{Var}(x) \leftrightharpoons \{x\}$, $\mathrm{FKonst}(x) \leftrightharpoons \emptyset$, $\mathrm{wst}(t) \leftrightharpoons 1$.

β) Für $x \in \mathrm{FKonst}^{(0,n)}(A)$ mit $n \in \mathbb{N}$ beliebig gilt:

$x \in \mathrm{Term}(A)$,

$\mathrm{Var}(x) \leftrightharpoons \emptyset$, $\mathrm{FKonst}(x) \leftrightharpoons \{x\}$, $\mathrm{wst}(x) \leftrightharpoons n$.

ii) Seien $t_1, t_2, \ldots, t_m \in \mathrm{Term}(A)$

α) Es gilt:

$$(t_1, t_2, \ldots, t_m) \in \mathrm{Term}(A),$$

$$\mathrm{Var}((t_1, t_2, \ldots, t_m)) \leftrightharpoons \bigcup_{1 \le i \le m} \mathrm{Var}(t_i),$$

$$\mathrm{FKonst}((t_1, t_2, \ldots, t_m)) \leftrightharpoons \bigcup_{1 \le i \le m} \mathrm{FKonst}(t_i)$$

$$\mathrm{wst}((t_1, t_2, \ldots, t_m)) \leftrightharpoons \sum_{i=1}^{m} \mathrm{wst}(t_i).$$

β) Für $f \in \mathrm{FKonst}^{(m',n')}(A)$ mit $m' = \sum_{i=1}^{m} \mathrm{wst}(t_i)$ gilt:

$$f(t_1, t_2, \ldots, t_m) \in \mathrm{Term}(A),$$

$$\mathrm{Var}(f(t_1, t_2, \ldots, t_m)) \leftrightharpoons \bigcup_{1 \le i \le m} \mathrm{Var}(t_i),$$

$$\mathrm{FKonst}(f(t_1, t_2, \ldots, t_m)) \leftrightharpoons \{f\} \cup \bigcup_{i=1}^{m} \mathrm{FKonst}(t_i).$$

$$\mathrm{wst}(f(t_1, t_2, \ldots, t_m)) \leftrightharpoons n'.$$

Bemerkung

Mithilfe metasprachlicher Abkürzungen und Konventionen (siehe z. B. Oberschelp [1974] S. 37) lassen sich den verschiedenen Anwenderklassen angepaßtere Notationen für Terme erzielen. Unsere Term-Notation hat hierbei in ähnlicher Weise eine Normalform-Funktion wie z. B. pränexe Normalformen in der Logik.

Die Definition der Variablenmenge, Funktionskonstantenmenge und Wertstelligkeit eines Terms dient später zur Feststellung (und dem Ausschluß) einer Teilklasse von "unsinnigen" Programmen.

Def. 8 (Interpretation und Wert von Termen)

Sei $t \in Term(A)$ beliebig.

a) J ordnet t ein m-Tupel von Bereichskonstanten $b(t,J) \in (BKonst(A))^m$ zu mit $m = wst(t)$, das sich folgendermaßen ergibt:

i) Falls $t \in Var(A)$, so sei $b(t,J) \leftrightharpoons (b(v,J_V))$.

ii) Falls $t \in FKonst^{(0,n)}(A)$, so sei $b(t,J) \leftrightharpoons (b(t,J_K))$.

iii) Falls $t = (t_1,t_2,\ldots,t_n)$, so sei $b(t,J) \leftrightharpoons b(t_1,J) \bullet \ldots \bullet b(t_n,J)$ [1].

---

1) $(b_1,b_2,\ldots,b_n) \bullet (b'_1,b'_2,\ldots,b'_{n'}) = (b_1,b_2,\ldots,b_n,b'_1,\ldots,b'_{n'})$ (Konkatenation von Wortfolgen).

iv) Gilt nicht i), ii) oder iii), so besitzt t die Form

$f(t_1,t_2,\ldots,t_{m'})$ mit $m'\geq 1$, $f \in FKonst^{(m_f,n_f)}(A)$

(wobei $m_f,n_f \in N$ durch f eindeutig bestimmt sind)

und $(\forall\ 1\leq i\leq m')\ t_i \in Term(A)$.

Ist f durch J_K das (m_f+n_f)-Tupel

$(b_1(f),b_2(f),\ldots,b_{m_f}(f),b_{m_f+1}(f),\ldots,b_{m_f+n_f}(f))$

zugeordnet, so gilt: $b(t,J) \leftrightharpoons (b_{m_f+1}(f),\ldots,b_{m_f+n_f}(f))$.

b) Sei zusätzlich $\sigma \in \Sigma_J$ beliebig.

$t^{J,\sigma}$ bezeichnet den Wert von t bezüglich der Interpretation J und Belegung σ, der sich folgendermaßen ergibt:

i) Falls $t \in Var(A)$: $t^{J,\sigma} \leftrightharpoons \sigma(t)$.

ii) Falls $t \in FKonst^{(0,n)}(A)$: $t^{J,\sigma} \leftrightharpoons w(t,J_K)$.

iii) Falls $t = (t_1,t_2,\ldots,t_n)$: $t^{J,\sigma} \leftrightharpoons (t_1^{J,\sigma},t_2^{J,\sigma},\ldots,t_n^{J,\sigma})$.

iv) Gilt nicht i), ii) oder iii), so besitzt t die Form

$f(t_1,t_2,\ldots,t_m)$ mit $m\geq 1$, $f \in FKonst^{(m_f,n_f)}(A)$ und

$(\forall\ 1\leq i\leq m)\ t_i \in Term(A)$, wobei gelte:

$$t^{J,\sigma} \leftrightharpoons \begin{cases} \Omega \text{ falls } \exists i(1\leq i\leq m \wedge t_i^{J,\sigma} = \Omega) \\ w(f,J)(t_1^{J,\sigma},t_2^{J,\sigma},\ldots,t_m^{J,\sigma}) \text{ sonst.} \end{cases}$$

Def. 9

$LZ \leftrightharpoons \{=, \perp, \top, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists\}$ ist eine Menge von logischen Zeichen.

Bemerkung

Zum Aufbau von Formeln benötigen wir neben Variablen, Termen und Hilfszeichen logische Zeichen.
Mit Ausnahme des Gleichheitszeichens "=" dienen die logischen Zeichen nur zur Beschreibung von Programmzuständen in Form von Zusicherungen, während die Funktionskonstanten eines Anwenderalphabets sowohl zur Beschreibung von Berechnungen als auch zur Beschreibung von Programmzuständen dienen werden.
Um beim Prozeß der Programmverifikation klar zwischen den logischen Operatoren der "Berechnungsebene" (Funktionskonstanten eines Anwenderalphabets) und den logischen Operatoren der "Verifikationsebene" (unseren definierten logischen Zeichen) unterscheiden zu können, gelte im folgenden für unser Anwenderalphabet A, daß $A \cap LZ = \emptyset$.

Def. 10

Die Menge der Formeln über A (kurz Form(A)) sowie die Menge der Funktionskonstanten FKonst(Φ), die Menge der freien Variablen FVar(Φ) und die Menge der gebundenen Variablen GVar(Φ) einer Formel $\Phi \in$ Form(A) sind induktiv definiert:

i) $\{\top, \perp\} \cup \{(t = t') \mid t, t' \in \mathrm{Term}(A) \wedge \mathrm{wst}(t) = \mathrm{wst}(t')\} \subseteq \mathrm{Form}(A)$,
$(\forall E \in \{\mathrm{FKonst}, \mathrm{FVar}, \mathrm{GVar}\})(E(\top) \leftrightharpoons \emptyset \wedge E(\perp) = \emptyset)$,
$\mathrm{FKonst}((t = t')) \leftrightharpoons \mathrm{FKonst}(t) \cup \mathrm{FKonst}(t')$,
$\mathrm{FVar}((t = t')) \leftrightharpoons \mathrm{Var}(t) \cup \mathrm{Var}(t'))$,
$\mathrm{GVar}((t = t')) \leftrightharpoons \emptyset$.

ii) Für $\Phi, \Phi' \in Form(A)$, $v \in Var(A)$ mit $v \notin GVar(\Phi)$ gilt:

$\neg \Phi$, $(\Phi \wedge \Phi')$, $(\Phi \vee \Phi')$, $(\Phi \rightarrow \Phi')$, $(\Phi \leftrightarrow \Phi')$,

$\forall v \Phi$, $\exists v \Phi$ sind Formeln über A,

$(\forall E \in \{FKonst, FVar, GVar\})$ $(E(\neg\Phi) \leftrightharpoons E(\Phi) \wedge$

$E((\Phi \wedge \Phi')) \leftrightharpoons E((\Phi \vee \Phi')) \leftrightharpoons E((\Phi \rightarrow \Phi')) \leftrightharpoons$

$E((\Phi \leftrightarrow \Phi')) \leftrightharpoons E(\Phi) \cup E(\Phi'))$,

$FKonst(\forall v\Phi) \leftrightharpoons FKonst(\exists v\Phi) \leftrightharpoons FKonst(\Phi)$,

$FVar(\forall v\Phi) \leftrightharpoons FVar(\exists v\Phi) \leftrightharpoons FVar(\Phi) \setminus \{v\}$,

$GVar(\forall v\Phi) \leftrightharpoons GVar(\exists v\Phi) \leftrightharpoons GVar(\Phi) \cup \{v\}$.

Bemerkung

Ähnlich wie die Variablen- und Funktionskonstantenmenge von Termen dienen auch die Funktionskonstantenmenge sowie die Menge der freien bzw. gebundenen Variablen von Formeln zum Ausschluß von Teilklassen "unsinniger" Programme und "unsinnig" sind nach unserem Verständnis nicht nur Programme mit unsinnig -definierten Berechnungen, sondern auch solche mit unsinnigen Spezifikationen (Zusicherungen).

Def. 11

Eine Formel der Form $(t = t')$ mit $t, t' \in Term(A)$ heißt Gleichung über A.

Bemerkung

Zur Vereinfachung unserer Theoriebildung wird sich die Form von Abfragen bei Programm-Verzweigungen auf Gleichungen beschränken.

Def. 12 (Wert von Formeln)

$\Phi^{J,\sigma}$ bezeichnet den Wert von $\Phi \in \mathrm{Form}(A)$ bezüglich der Interpretation J und der Belegung $\sigma \in \Sigma_J$, der sich folgendermaßen ergibt:

i) $\top^{J,\sigma} \leftrightharpoons F, \quad \bot^{J,\sigma} \leftrightharpoons W$ [1)]

$$(t = t')^{J,\sigma} \leftrightharpoons \begin{cases} W, \text{ falls } t^{J,\sigma} \neq \Omega \wedge t'^{J,\sigma} \neq \Omega \wedge t^{J,\sigma} = t'^{J,\sigma}. \\ F, \text{ falls } t^{J,\sigma} \neq \Omega \wedge t'^{J,\sigma} \neq \Omega \wedge t^{J,\sigma} \neq t'^{J,\sigma}. \\ \Omega \text{ sonst}. \end{cases}$$

für $t, t' \in \mathrm{Term}(A)$ mit $\mathrm{wst}(t) = \mathrm{wst}(t')$ beliebig.

ii)

$$(\neg\psi)^{J,\sigma} \leftrightharpoons \begin{cases} W, \text{ falls } \psi^{J,\sigma} = F. \\ F, \text{ falls } \psi^{J,\sigma} = W. \\ \Omega \text{ sonst}. \end{cases}$$

$$(\psi \wedge \psi')^{J,\sigma} \leftrightharpoons \begin{cases} W, \text{ falls } \psi^{J,\sigma} = \psi'^{J,\sigma} = W. \\ F, \text{ falls } \psi^{J,\sigma} \neq \Omega \wedge \psi'^{J,\sigma} \neq \Omega \wedge \\ \qquad (\psi^{J,\sigma} = F \vee \psi'^{J,\sigma} = F). \\ \Omega \text{ sonst}. \end{cases}$$

$$(\psi \vee \psi')^{J,\sigma} \leftrightharpoons (\neg(\neg\psi \wedge \neg\psi'))^{J,\sigma}.$$

$$(\psi \rightarrow \psi')^{J,\sigma} \leftrightharpoons (\neg\psi \vee \psi')^{J,\sigma}.$$

$$(\psi \leftrightarrow \psi')^{J,\sigma} \leftrightharpoons ((\psi \rightarrow \psi') \wedge (\psi' \rightarrow \psi))^{J,\sigma}.$$

$$(\forall v\ \psi)^{J,\sigma} \leftrightharpoons \begin{cases} W, \text{ falls für alle } x \in J_K(b(v,J_K))\ \psi^{J,\sigma^v_x} = W. \\ \Omega, \text{ falls für ein } x \in J_K(b(v,J_K))\ \psi^{J,\sigma^v_x} = \Omega. \\ F \text{ sonst}. \end{cases}$$

$$(\exists v\ \psi)^{J,\sigma} \leftrightharpoons (\neg\ \forall v\ \psi)^{J,\sigma}$$

für $\psi \in \mathrm{Form}(A)$ und $v \in \mathrm{Var}(A)$ beliebig.

Bemerkung

σ^v_x ist diejenige Belegung, die v mit x belegt und sonst mit σ übereinstimmt.

---

1) Als Wahrheitswerte benutzen wir "W" für wahr und "F" für falsch.

<u>Beispiele</u>

Für ein Anwenderalphabet A:

$BKonst(A) \leftrightharpoons \{integer, boolean\}$

$FKonst^{(0,1)}(A) \leftrightharpoons \{"0", "1", max, \underline{true}, \underline{false}\}$

$FKonst^{(1,1)}(A) \leftrightharpoons \{halb, dop, odd\}$

$FKonst^{(2,1)}(A) \leftrightharpoons \{sub, add, grgl\}$

$FKonst^{(m,n)}(A) \leftrightharpoons \emptyset$ für $m>2 \vee n>1$

$Var(A) \leftrightharpoons \{v_1, v_2, v_3, \ldots\}$

Für eine Konstanten-Interpretation J_K von A:

$J_K(integer) \leftrightharpoons \{x \mid x \in N_0 \wedge x \leq 2^{48}\}$, $J_K(boolean) \leftrightharpoons \{true, false\}$,

$b("0", J_K) = b("1", J_K) = b(max, J_K) \leftrightharpoons (integer)$

$w("0", J_K) \leftrightharpoons 0$, $w("1", J_K) \leftrightharpoons 1$, $w(max, J_K) \leftrightharpoons 2^{48}$

$b(\underline{true}, J_K) = b(\underline{false}, J_K) \leftrightharpoons (boolean)$

$w(\underline{true}, J_K) \leftrightharpoons true$, $w(\underline{false}, J_K) \leftrightharpoons false$

$b(halb, J_K) = b(dop, J_K) \leftrightharpoons (integer, integer)$

$$w(halb, J_K) \leftrightharpoons \left\langle x \mapsto \begin{cases} \frac{x}{2}, & \text{falls } x \text{ gerade} \vee x = 0 \\ \frac{x-1}{2} & \text{sonst} \end{cases} \mid x \in J_K(integer) \right\rangle$$

$$w(dop, J_K) \leftrightharpoons \left\langle x \mapsto \begin{cases} 2 \cdot x, & \text{falls } x \leq 2^{47} \\ \Omega & \text{sonst} \end{cases} \mid x \in J_K(integer) \right\rangle$$

$b(odd, J_K) \leftrightharpoons (integer, boolean)$

$$w(odd, J_K) \leftrightharpoons \left\langle x \mapsto \begin{cases} true, & \text{falls } x \text{ ungerade} \\ false & \text{sonst} \end{cases} \mid x \in J_K(integer) \right\rangle$$

$b(sub, J_K) = b(add, J_K) \leftrightharpoons (integer, integer, integer)$

$$w(sub, J_K) \leftrightharpoons \left\langle (x,y) \mapsto \begin{cases} x-y, & \text{falls } x \geq y \\ \Omega & \text{sonst} \end{cases} \mid x, y \in J_K(integer) \right\rangle$$

$$w(add,J_K) \leftrightharpoons \left\langle (x,y) \mapsto \begin{cases} x+y, \text{ falls } x+y \leq 2^{48} \\ \Omega \text{ sonst} \end{cases} \middle| x,y \in J_K(integer) \right\rangle$$

$$b(grgl,J_K) \leftrightharpoons (integer,integer,boolean)$$

$$w(grgl,J_K) \leftrightharpoons \left\langle (x,y) \mapsto \begin{cases} true, \text{ falls } x \geq y \\ false \text{ sonst} \end{cases} \middle| x,y \in J_K(integer) \right\rangle$$

Für eine Variablen-Interpretation J_V von A:

$b(v,J_V) \leftrightharpoons$ integer für $v \in Var(A)$ beliebig.

Für Terme über A:

"0", v_3, (0,1,max), $halb(v_3)$, $dop(halb(v_3))$,

$wst(0) = wst(v_3) = 1$, $wst(dop(halb(v_3))) = 1$,

$FKonst(dop(halb(v_3))) = \{halb,dop\}$.

Für Formeln über A:

$(dop(halb(v_3)) = v_4)$, $((v_1 = v_4) \wedge (grgl(halb(v_1),v_3) = \underline{true}))$

$\exists v_1(v_1 = v_4)$,

$FVar(dop(halb(v_3)) = v_4) = \{v_3,v_4\}$

$GVar(\exists v_1(v_1 = v_4)) = \{v_1\}$

$FVar(\exists v_1(v_1 = v_4)) = \{v_4\}$

Für Belegungen über (J_K,J_V):

$$\sigma_1 \leftrightharpoons \left\langle v \mapsto \begin{cases} 2, \text{ falls } v = v_1 \\ 4, \text{ falls } v = v_2 \\ \Omega \text{ sonst} \end{cases} \middle| v \in Var(A) \right\rangle$$

$$\sigma_2 \leftrightharpoons \left\langle v \mapsto \begin{cases} 2 \cdot \sigma_1(v_1), \text{ falls } v = v_1 \\ \frac{\sigma_1(v_2)}{2}, \text{ falls } v = v_2 \\ \Omega \text{ sonst} \end{cases} \middle| v \in Var(A) \right\rangle$$

Für Werte von Termen und Formeln:

$$\mathrm{dop}(v_1)^{J,\sigma_1} = 4$$

$$v_2^{J,\sigma_2} = 2 \quad ,$$

$$(\ \mathrm{halb}(v_1) = v_2)^{J,\sigma_1} = F$$

$$\exists v_3 (v_3 = v_2)^{J,\sigma_1} = W$$

3. Eine auf Anwenderbereichen aufbauende Modellbildung von Rechensystemen

3.1 Algorithmische Modellbildung

3.1.1 Pragmatik, Syntax und Semantik "von-Neumann-ähnlicher" Sprachen

3.1.1.1 Pragmatische Vorüberlegungen zur Klassifizierung und Strukturierung von Programm-Bausteinen

Eine klare, transparente, leicht verständliche Gliederung und Strukturierung von Rechenalgorithmen erleichtert nicht nur den Nachweis der Korrektheit (bzw. Zuverlässigkeit), sondern gegebenenfalls auch die Fehlersuche und Fehlerkorrektur. Daneben soll die Gliederung und Strukturierung eines Algorithmus auch gewährleisten, daß lokale Änderungen am Algorithmus auf die Umgebung nur einen geringen, leicht überschaubaren Einfluß haben ("Stabilität", "lokale Änderungsfreundlichkeit") und kleine Modifikationen der Aufgabenstellung auch nur kleine Änderungen am Algorithmus implizieren ("globale Änderungsfreundlichkeit", "Erweiterungsfähigkeit").

Ein erstes, wesentliches Hilfsmittel für eine derart gute Gliederung und Strukturierung eines Rechenalgorithmus ist seine Modularisierung, wobei wir im Unterschied zur Literatur (siehe z.B. Balzert [1977], Altmann [1978]) alle aus der Sicht des Anwenders bedeutungsvollen Bausteine für Rechenalgorithmen, die genau einen Ein- und Ausgang sowie mindestens eine Wertzuweisung besitzen, unter dem Begriff Modul zusammenfassen. Nur die Teilklasse der vollständig spezifizierten Moduln wird weitgehend die in der Literatur postulierten Eigenschaften von "Programm-Moduln" erfüllen.

Weitere Hilfsmittel für eine gute Gliederung und Strukturierung von Rechenalgorithmen ergeben sich zum einen aus der Frage nach geeigneten Verknüpfungsstrukturen für die Moduln von Rechenalgorithmen und zum anderen aus der Frage nach einer klaren, transparenten, leicht verständlichen Gliederung und Strukturierung der Moduln selbst, wobei sich schon vorneweg sagen läßt, daß ohne eine Normierung - sowohl der Verknüpfungsstrukturen von

Moduln als auch der Modulstrukturen selbst - sich insbesondere die Forderung nach leicht verständlichen Rechenalgorithmen nicht realisieren läßt (siehe z.B. auch Winkler [1977]).
Ohne die Verwendung normierter Bausteine ("Moduln") und normierter "Spielregeln" läßt sich bestenfalls schlecht wissenschaftlich argumentieren.

Betrachten wir zunächst die Frage nach geeigneten Modul-Verknüpfungsstrukturen. Nach Simon [1969] ist die Hierarchisierung ein zentrales Strukturierungs-Schema für den Aufbau komplexer Systeme und es ist naheliegend, daß unsere Modulverknüpfungsstrukturen den Aufbau hierarchischer Modul-Systeme ermöglichen sollten. Neben der "sequentiellen" Verknüpfung von Moduln (Modulkonkatenation) benötigen wir also noch eine "hierarchische" Verknüpfung von Moduln (Modulschachtelung).
Im Unterschied zu Programmiersprachen wie z. B. ALGOL 60 werden wir Modulschachtelungen jedoch nur über Prozeduraufrufe ermöglichen. Diese Einschränkung vereinfacht nicht nur die Struktur unserer Programm-Moduln und damit die darauf aufbauende Theorie der Semantik und Verifikation, sondern die dadurch erzwungene Programm-Lokalität[1] und Programm-Modularisierung dürfte auch für die Praxis des Programmierens vorteilhaft sein.
Strebt man eine Liberalisierung der restriktiven Handhabung von Modulschachtelungen an, so fällt es schwer hierfür ein geeignetes Maß zu finden: Mit einer normierten oberen Schranke $n \in N$ $(n=2,3,..?)$ für die zulässige Tiefe nicht-prozeduraler Modul-Schachtelungen läßt sich das Ziel des Ausschlusses unübersichtlicher, schwierig handhabbarer Programm-Moduln nicht erreichen (wie man sich anhand

---

1) "By locality, we mean that property that obtains when all relevant parts of a program are found on the same place - on the same page, perhaps. Without good locality, a programmer working with the listing of a program has to be turning pages constantly unless his syntactic memory is sufficiently good to remember the material on the other pages that is relevant to the current page" (Weinberg [1971] pp. 229).

von Beispielen leicht klarmacht). Eine absolute Beschränkung der Modulgröße, z. B. auf ein x-faches der Bildgröße von Terminals, erscheint dafür schon besser geeignet - wenn man Fragen nach einem angemessenen theoretischen Modell unterdrückt. Eine prinzipiell andere Möglichkeit zur Verminderung des Zwangs zu Modulschachtelungen über Prozeduraufrufe besteht darin, die Struktur häufig vorkommender Modulschachtelungen zu normieren und in Form ausgezeichneter Modulklassen dem Programmierer zur Verfügung zu stellen. Doch auch dieser Ansatz stellt nur einen partiell zufriedenstellenden Kompromiß dar.

Es gibt somit mehrere Gründe dafür, die Schachtelung von Moduln nur über Prozeduraufrufe zu ermöglichen. Einen potentiell möglichen Nachteil aus dieser Vorgehensweise, der aus der Verwendung von Parametern bei Prozeduraufrufen resultiert, können wir dabei dadurch vermeiden, daß wir neben den üblichen umgebungsunabhängig definierten globalen Prozeduren mit Parametern eine zweite Prozedurklasse umgebungsabhängig definierter lokaler Prozeduren ohne Parameter verwenden. Die aus der Parameterisierung von Prozeduraufrufen resultierende Mehrarbeit (z. B. Beschreibung der Parameter) bei der Programmierung und Programmwartung wird also vermieden. Daneben lehnt sich die Unterscheidung zwischen globalen und lokalen Prozeduren an eine Unterscheidung von Simon [1969] an[1], wo in hierarchischen Systemen zwischen den Interaktionen unter Teilsystemen ("globalen Prozeduren") und den Interaktionen innerhalb eines Teilsystems ("lokalen Prozeduren") unterschieden wird.

Auf Inkarnationsaufrufe[2] innerhalb anwenderorientierter Rechenprogramme wird verzichtet, da eine Entscheidung zwischen Prozedur-

---

1) Simon [1969] pp. 106:"... We have seen, that hierarchies have the property of near decomposability. Intracomponent linkages are generally stronger than intercomponent linkages...."

2) Die Bezeichnung "Inkarnationsaufruf" erscheint angemessener als die Bezeichnung "Makroaufruf".

und Inkarnationsaufrufen u.a. auch von der Struktur der verwendeten Rechenmaschine, wie z.B. Zahl und Art der Prozessoren und Größe des Programm-Speichers, d.h. von technologie-abhängigen Parametern abhängig ist und sich Modul-Inkarnationen unter Berücksichtigung technologie-spezifischer Parameter auch noch bei der Programmübersetzung aus Prozeduraufrufen erzeugen lassen, d.h. die Beschränkung auf Prozeduraufrufe nicht automatisch einen hohen Datentransfer (zwischen Moduln) bei der Programmausführung implizieren muß.

Mit dem Verzicht auf Inkarnationsaufrufe erübrigt sich auch die Frage nach der Zweckmäßigkeit ALGOL-ähnlicher Blockstrukturen zur Modul-Verschachtelung, denn Inkarnationsaufrufe besitzen gegenüber Blockstrukturen die folgenden Vorteile:

- Der geringere Platzbedarf von Inkarnationsaufrufen führt zu kompakterer Darstellung der auf gleichem Verfeinerungsniveau (Anwenderbereich) liegenden Programmteile, d.h. zu einer besseren Programm-Lokalität.
- Inkarnationsaufrufe bieten bessere Ansätze fürs "Information-Hiding", da Moduln unterschiedlicher Verfeinerungsniveaus auch räumlich leicht trennbar sind.
- Mittels Inkarnationsaufrufen verknüpfte Moduln können räumlich in verschiedenen Lochkartenstapeln bzw. auf verschiedenen Dateien untergebracht sein und sind damit relativ leicht getrennt übersetzbar und wartbar.

Die Möglichkeit zu rekursiver Modulschachtelung mittels rekursiv verschachtelten Prozeduraufrufen wird hierbei ausgeklammert werden: Entsprechend der Einschränkung unserer Theoriebildung auf Rechensysteme für nicht-rekursive Algorithmen werden wir nur hierachisch verschachtelte Prozeduraufrufe erlauben.

Der Verzicht auf Funktionsprozeduren ist durch den normierten Bezug eines Moduls auf einen jeweils fest vorgegebenen Anwenderbereich nahegelegt: Wegen der Bedeutung von Anwenderbereichen im Rahmen unserer Theoriebildung sollte der Übergang zu einem anderen Anwenderbereich nicht implizit ("versteckt") mithilfe von Funktionsprozeduren erfolgen können. Die Zweckmäßigkeit von Funktionsprozeduren zeigt sich im Rahmen unserer Theoriebildung im wesentlichen erst bei der Frage nach der Implementierung von Anwenderbereichen.

Bei der Frage nach einer zweckmäßigen Gliederung und Strukturierung des Modulaufbaus unterscheiden wir zwischen der Form der Modulspezifikation ("äußerer Struktur") und der Form der Modulberechnungen ("innerer Struktur").
Entsprechend der Form der Modulspezifikation unterscheiden wir zunächst zwischen partiell spezifizierten und vollständig spezifizierten Moduln[1], wobei wir bei den partiell spezifizierten Moduln wiederum unterscheiden zwischen aufrufbaren und nicht-aufrufbaren Moduln. Bei den nicht-aufrufbaren Moduln unterscheiden wir schließlich noch zwischen funktions-spezifizierten, kommentierten und nackten Moduln.

Die Verwendung vollständig spezifizierter Moduln, bei denen neben den durch den Modul realisierten Funktionen (in Form von Anfangs- und Endzusicherungen) auch noch benutzte Anwenderbereiche, globale und lokale Variablen und benutzte globale Prozeduren spezifiziert sind, bietet nicht nur Vorteile für den Programmentwurf (siehe Methodiken des Problemlösens), sondern ermöglicht bzw. erleichtert auch das getrennte Übersetzen und ggf. Ausführen von Programm-Moduln.

Die Forderung nach "intellektueller Bewältigbarkeit" von Moduln impliziert dabei u.a. auch eine Beschränkung der Anzahl der verwendeten Modulvariablen und der "Komplexität benutzter Anwenderbereiche" (Anzahl und/oder "Schwierigkeit" der vorkommenden Begriffe?) Diese Schranken werden zwar je nach "Geschmack" ver-

---

1) Vollständig spezifizierte Moduln und globale Prozeduren sind identisch.

schieden hoch angesetzt werden, wobei eine evtl. Normierung in den Bereich einer "Psychologie des Programmierens/Problemlösens" fällt (z.B. Dörner [1976]), die Zweckmäßigkeit einer Beschränkung der jeweils relevanten Information beim Programmieren (Problemlösen) dürfte jedoch nicht bestritten werden (siehe auch Weinberg [1971]), wobei sich über die Verwendung von Prozeduren derartige Schranken auch weitgehend realisieren lassen.

Neben den aufrufbaren vollständig spezifizierten Moduln wollen wir auch aufrufbare partiell spezifizierte Moduln zulassen, die nur innerhalb eines feststehenden Kontextes aufrufbar sind und deren Spezifikation aus pragmatischen Gründen (Erleichterung von Schreibarbeit, Straffung des Programmtextes) partiell kontextabhängig erfolgen kann und im wesentlichen auf die Spezifikation der durch den Modul realisierten Funktionen beschränkt wird.

Bei nichtaufrufbaren Moduln, die innerhalb anderer Moduln statisch eingebettet sind und die auf verschiedenen Betrachtungsebenen (z.B. der der Verifikation) nicht als (partiell-)umgebungsunabhängige Moduln fungieren müssen, erübrigt sich ein Zwang zur Spezifikation der durch den Modul realisierten Funktionen: Bei derartigen Moduln kann eine Spezifikation vollständig fehlen (nackte Moduln), eine Spezifikation umgangssprachlich vorliegen (kommentierte Moduln) oder eine Spezifikation formalisiert in Form von Zusicherungen gegeben sein (funktions-spezifizierte Moduln), wobei letzteres den Entwurfsprozeß erleichtert[1)] und zu einer feineren Modularisierung der Programm-Verifikation und damit zu einer feineren Programm-Dokumentation führt. Eine formale Spezifikation mittels Zusicherungen wird dabei stets durch eine umgangssprachliche Kommentierung (einen "Modulkopf") ergänzt. Umgangssprachliche Kommentierungen des Modulaufbaus, der Modulfunktion, des zugrundeliegenden "Modulplans", der verwendeten Anwenderalphabete u. ä. werden mit der Beschränkung von Kommentar auf den Modulanfang soweit normiert, daß Kommentierungen der Form "erhöhe Zähler um 1" weitgehend ausgeschlossen sind.

---

1) Simon [1969] zeigt, daß der Aufbau von komplexen Systemen aus einfacheren Systemen sehr viel schneller möglich ist, wenn fest vorgegebene "Zwischenformen" existieren. Der formalen Beschreibung derartiger "Zwischenformen" dienen bei unserer algorithmischen Modellbildung u. a. auch die Zusicherungen.

Zwischen den aufrufbaren, vollständig spezifizierten Moduln und den aufrufbaren partiell spezifizierten Moduln ließen sich entsprechend dem Grad der Umgebungsunabhängigkeit von Moduln noch die folgenden zwei ausgezeichneten Zwischenstufen einführen:

a) Die Vereinbarung der verwendeten Variablen erfolgt weitestgehend umgebungsunabhängig innerhalb des Moduls, d.h. allen im Modul verwendeten Variablen wird (zweckmäßigerweise an einer ausgezeichneten Stelle im "Modulkopf") ein Bereich zugeordnet, wobei zwischen globalen und lokalen Variablen zu unterscheiden ist. Die globalen Variablen sind dabei nur insoweit umgebungsabhängig, als zwischen den aktuellen Parametern eines Modulaufrufs und den Anfangs- und Endwerten der globalen Variablen ein normierter Zusammenhang erwartet wird (ALGOL 60 - Prozeduren mir formalen Parametern und Hilfsvariablen).

b) Die Variablen sind bis auf einige, die als lokale Variable ausgezeichnet sind , in einem übergeordneten Modul vereinbart (ALGOL 60 - Prozeduren ohne Parameter mit Hilfsvariablen; ALGOL 60 - Blockkonzept, wenn "Schachtelung von Blöcken" durch "Aufruf von Blöcken" ersetzt wird).

Auf Moduln der ersten Zwischenstufe wollen wir deshalb verzichten, weil derartige Moduln relativ einfach, d.h. ohne viel Schreibarbeit[1], in vollständig spezifizierte Moduln transformierbar sind und der geringe Mehraufwand hierfür im Hinblick auf die daraus resultierenden Vorteile - nicht zuletzt auch in Form einer vereinfachten Theoriebildung - akzeptiert werden sollten.

Bei Moduln der zweiten Zwischenstufe läßt sich kein akzeptierbarer Zweck angeben, dem die Auszeichnung lokaler Variablen dienen könnte, während ein Verzicht auf derartige Moduln nicht nur zu einer Vereinfachung unserer Theoriebildung führt, sondern durch die Konzentration der Variablenvereinbarungen an ausgezeichneten Programmpunkten (im Kopf vollständig spezifiz.

---

1) Die Kennzeichnung des Anwendungsbereichs erfolgt hierbei einfach durch einen Namen.

Moduln) vor allem auch die Anpassung des Programms an modifizierte Aufgabenstellungen erleichtert. Die Verwendung des ALGOL 60-Blockkonzepts und von ALGOL 60-Prozeduren ohne Parameter lassen sich somit höchstens aus Überlegungen zur (technologie-abhängigen) maschinellen Programm-Effizienz begründen.

Neben der Normierung (Standardisierung) der Form möglicher Modulspezifikationen ist auch eine Normierung der Form von Modulberechnungen zweckmäßig, denn wie schon erwähnt ist die Verwendung normierter, standardisierter Begriffsbildungen ("Bausteine") eine notwendige Voraussetzung wiss. Argumentation und ein Programm sollte ja nicht nur eine Anweisungsfolge für einen Rechner beschreiben, sondern sollte auch eine wissenschaftliche Argumentation dafür sein, daß eine korrekte Programm-Ausführung stets zu korrekten, erwarteten Ergebnissen führt. Die Verwendung beliebig strukturierter Modulberechnungen - und damit die beliebige Verwendung von goto's - scheidet somit aus.

Unsere Normierung der Form von Modulberechnungen unterscheidet dabei zunächst zwischen einfach-seriellen Moduln einerseits und nicht einfach-seriellen Moduln andererseits. Einfach-serielle Moduln beschreiben Wertzuweisungen, die entweder mittels einer "Assignation" direkt oder mittels eines lokalen Prozeduraufrufs (ohne Parameter) bzw. eines globalen Prozeduraufrufs (mit Parameter) indirekt erfolgen können, wobei die Bedeutung dieser Wertzuweisungen auch bei Prozeduraufrufen direkt angegeben ist, d.h. nicht unter Rückgriff auf die aufzurufende Prozedur erst ermittelt werden muß. Diese explizite Beschreibung der Bedeutung von Prozeduraufrufen direkt am Aufrufpunkt verbessert nicht nur die Programmdokumentation und erleichtert damit die Programmverifikation und Programmwartung, sondern vereinfacht auch in erheblichem Maße unsere Theoriebildung für Rechenprogramme.

Die Beschränkung auf deterministische Wertzuweisungen, bei denen die den Programmvariablen zuzuordnenden Werte jeweils eindeutig bestimmt sind, sowie der Wunsch, die Bedeutung von Prozeduraufrufen mittels derartiger Wertzuweisungen dokumentieren zu können, führt dazu, daß die Endzusicherung aufrufbarer Moduln eine deterministische Wertzuweisung über einem geeignet gewählten Anwender-

alphabet sein muß. Trotz der großen praktischen Bedeutung dieser Einschränkung bzw. dem großen Nutzen, den die Verwendung nicht-deterministischer Algorithmen fürs Programmieren bietet (Dijkstra [1977]), mußte auf die Erweiterung unserer Theoriebildung um nicht-deterministische Wertzuweisungen und damit zusammenhängende Fragestellungen verzichtet werden.

Wegen ihrer einfachen Struktur wurde die serielle Verknüpfung (Konkateration) von Wertzuweisungen noch zur Klasse der einfachen Moduln hinzugenommen.

Nicht einfach-serielle Moduln ergeben sich aus der Zusammenfassung einfach-serieller Moduln entsprechend vorzugebender einheitlicher Strukturmerkmale, die einen (statischen) Modulaufbau ermöglichen sollen, der dem gewünschten (dynamischen) Berechnungsprozeß möglichst gut angepaßt ist.

Den Widerspruch zwischen dem Wunsch nach möglichst wenigen unterschiedlichen Modulverknüpfungs-Strukturen einerseits - zur Erleichterung der praktischen Handhabung und Theoriebildung - und dem nach einer möglichst großen Zahl verfügbarer unterschiedlicher Modulverknüpfungs-Strukturen andererseits - zur möglichst "ungekünstelten" Darstellung gewünschter Berechnungsprozesse - wollen wir hier jedoch nicht systematisch ausbreiten (siehe allgemeine goto-Diskussion, z.B. in Knuth [1974]). Unsere gewählten Modulverknüpfungs-Strukturen sind jedoch nicht willkürlich gewählt, sondern versuchen diesem Widerspruch weitgehend gerecht zu werden, wobei der restriktive Standpunkt im Hinblick auf eine einfachere Theoriebildung stärker gewichtet wurde.

Eine zweite pragmatisch begründete Forderung an Modulverknüpfungsstrukturen, die wir im folgenden erfüllen wollen, ist die nach weitestgehend umgebungsunabhängiger Bedeutung ihrer Komponenten[1], womit zum einen die Programm-Lokalität gesteigert, zum anderen die Semantik-Definition vereinfacht wird und beide Effekte nicht zuletzt die Erlernbarkeit und Handhabbarkeit einer Programmiersprache - und damit die Reduktion von Programmierfehlern - erleichtern.

---

1) Z.B der Berechnungssymbole.

3.1.1.2 Syntax

Def. 1

Die Menge der Kontrollzeichen KZ besteht aus

- den Berechnungssymbolen
  assi, ⇒, lcall, gcall, =,
  para, if, case, exit, repeat, fini, one,
  loop, otherwise, end, p-end, c-, p- ,
- den Spezifikationssymbolen
  lproc, gproc, head, b-dom, v-dom, input, output,
  global, local, bounded, type, ref, res, {, }, frei,
- den Hilfssymbolen
  "(", ")", ":" und ",".

Bemerkung

Die Berechnungssymbole dienen zunächst zur leicht einprägsamen, übersichtlichen Kennzeichnung der Form von Modulberechnungen, d. h. zur Unterscheidung zwischen Assignationen, lokalen und globalen Prozeduraufrufen und den verschiedenen Klassen nicht-einfacher Moduln.
Neben diesem am Bezugspunkt Mensch bzw. Programmierer orientierten Zweck dienen sie jedoch auch zur Definition von Berechnungen, d. h. zur Präzisierung einer operationellen Semantik bezüglich dem zugrundegelegten abstrakten Maschinenmodell. Dabei kann bezüglich des Einflusses auf den Kontrollfluß zwischen "aktiven" Berechnungssymbolen (z. B. para, if, case, exit) und "passiven" Berechnungssymbolen (z. B. die Sprungmarken end, otherwise) unterschieden werden.
Die Spezifikationssymbole dienen zur Unterscheidung zwischen partiell und vollständig spezifizierten Moduln, sowie zur Kennzeichnung kommentierter, funktions-spezifizierter und aufrufbarer Moduln, zur Kennzeichnung benutzter Anwenderbereiche, globaler und lokaler Variablen und benutzter Prozeduren sowie zur Kennzeichnung der Bedeutung von Prozeduraufrufen und parallelen Moduln. Das Zeichen frei dient der Freigabe nicht mehr benötigter lokaler Variablen.
Zur Darstellung von Listen werden noch die Hilfssymbole "(", ")" und "," benötigt; das Zeichen ":" dient als (redundantes) Trennzeichen hinter Marken.

Def. 2

$\mathfrak{A}$ bezeichne im folgenden eine beliebige endliche Menge von Anwenderbereichen, wobei für alle $(A,J) \in \mathfrak{A}$ gelte:

$(KZ \cup LZ \cup HZ) \cap A = \emptyset$.

Jedem $(A,J) \in \mathfrak{A}$ sei eine Menge NNam(A) von Namen für natürliche Zahlen mit $NNam(A) \cap (KZ \cup LZ) = \emptyset$, eine bijektive Funktion $fN(A) : NNam(A) \xrightarrow{bij} N$, sowie eine Menge MNam(A) von Modulnamen mit $MNam(A) \cap (KZ \cup LZ \cup A) = \emptyset$ zugeordnet.

BNam($\mathfrak{A}$) bezeichne eine Menge von Bereichs-Namen mit $BNam(\mathfrak{A}) \cap KZ = \emptyset$, wobei jedem Bereichs-Namen aus BNam($\mathfrak{A}$) ein Anwenderbereich aus $\mathfrak{A}$ zugeordnet sei.
K bezeichne ein beliebiges Kommentar-Alphabet, für das gelte:
$K \cap KZ = \emptyset$.
A sei das Anwenderalphabet eines beliebigen Anwendungsbereichs $(A,J) \in \mathfrak{A}$.

Bemerkung

BNam($\mathfrak{A}$), NNam(A) und MNam(A) müssen nicht paarweise disjunkt sein, denn die gewünschte eindeutige Zuordnung eines Namens zu einer dieser Mengen läßt sich mithilfe eines einfachen Kontextes herstellen.

Unabhängig davon, ob in einem Anwenderalphabet A Namen (nullstellige Funktionskonstanten) für natürliche Zahlen vorhanden sind oder nicht, werden diese durch NNam(A) zur Kennzeichnung von Sprungmarken zur Verfügung gestellt.
Damit kann sich die Verwendung von natürlichen Zahlen auf zwei verschiedene Ebenen der Programmbearbeitung beziehen: Einmal auf die Speicher- und Operationsstruktur des einem Anwenderalphabet zugeordneten Rechenwerks und zum anderen auf das Programm-Steuerwerk, das die Programmbearbeitung steuert.

Def. 3

Seien $M_1, M_2, \ldots, M_n$ Mengen.

$Disj(M_i, 1 \leq i \leq n) \leftrightarrows$

$(\forall\ 1 \leq i, j \leq n)(i \neq j \rightarrow M_i \cap M_j = \emptyset)$

heißt Disjunktheitsbedingung für $M_1, M_2, \ldots, M_n$.

Bemerkung

Die abgekürzte Notation für die Disjunktheitsbedingung von Mengen vereinfacht nachfolgende Definitionen.

Def. 4

Seien $M_1, M_2, \ldots, M_n$ beliebige Tripel-Mengen.

$Konsi(M_i, 1 \leq i \leq n) \leftrightarrows$

$(\forall\ 1 \leq i, j \leq n)(\forall(name, est, ast) \in M_i)(\forall(name', est', ast') \in M_j)$

$(name = name' \rightarrow est = est' \wedge ast = ast')$

heißt Konsistenzbedingung bezüglich $M_1, \ldots, M_n$.

Bemerkung

Die Konsistenzbedingung von Tripel-Mengen wird sich im folgenden auf die Mengen $GPA(M_i)$ von Moduln $M_1, \ldots, M_n$ beziehen, d. h. es handelt sich um eine Konsistenzbedingung bezüglich der in Moduln $M_1, \ldots, M_n$ vorkommenden globelen Prozeduraufrufe. Analog zur Disjunktheitsbedingung von Mengen vereinfacht diese Konsistenzbedingung globaler Prozeduraufrufe nachfolgende Definitionen.

Def. 5

Eine Wertzuweisung w über A ist eine Zeichenkette der Form

$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),(t_2 \Rightarrow (v_{21},\ldots,v_{2n_2})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$

für die gilt:

$$(\forall\ 1 \le i \le m)\ (t_i \in \mathrm{Term}(A) \cup \{\underline{\mathrm{frei}}\} \wedge (t_i \in \mathrm{Term}(A) \rightarrow \mathrm{wst}(t_i) = n_i) \wedge$$
$$(\forall\ 1 \le j \le n_i)\ v_{ij} \in \mathrm{Var}(A) \wedge \mathrm{Disj}(\{v_{ij}\}, 1 \le j \le n_i)) \wedge$$

$$\mathrm{Disj}(\{v_{ik} \mid 1 \le k \le n_i\}, 1 \le i \le m).$$

Bezeichnungen

W(A) sei die Menge aller Wertzuweisungen über A

$$\mathrm{MVar}(w) \leftrightharpoons \bigcup_{i=1}^{m} \{v_{ik} \mid 1 \le k \le n_i\}$$

heißt Menge der modifizierbaren Variablen von w

$$\mathrm{Var}(w) \leftrightharpoons \mathrm{MVar}(w) \cup \bigcup_{i=1}^{m} \mathrm{Var}(t_i)$$

heißt Menge der Variablen von w

$$\mathrm{FKonst}(w) \leftrightharpoons \bigcup_{i=1}^{m} \mathrm{FKonst}(t_i)$$

heißt Menge der Funktionskonstanten von w

Bemerkung

Die Forderung nach leichter Programmlesbarkeit spricht für eine Darstellungsform, bei der direkt ersichtlich ist, welche Eingangsparameter (Terme t_i) welchen Ausgangsparametern (Variablen $v_{i1}, v_{i2}, \ldots, v_{in_i}$) zugeordnet sind, d. h. bei der dieser Zusammenhang nicht erst durch Abzählen von Term- und Variablenlisten ermittelt werden muß (wie z. B. bei der Darstellungsform $((t_1, t_2, \ldots, t_m) \Rightarrow (v_1, \ldots, v_n))$).

Gegenüber der vertrauteren Darstellungsform

$$(v_{i1}, v_{i2}, \ldots, v_{in_i}) := t_i$$

wird in unserer Darstellungsform, entsprechend der besonderen Bedeutung von Wertzuweisungen, auch ein besonderes, diese Bedeutung besser darstellendes (redundantes) Zeichen $\Rightarrow$ verwendet und entsprechend dem Lesen von links nach rechts werden zuerst die Eingangs- und dann erst die Ausgangsparameter aufgelistet.

Die explizite Freigabe nicht mehr benötigter Variablenwerte mithilfe von "(<u>frei</u> $\Rightarrow$ $(v_{i1}, \ldots, v_{in_i})$)", wobei der Wert von v_{ij} für $1 \leq j \leq n_i$ nach dieser Freigabe undefiniert ist, erleichtert sowohl die problem- als auch die technologieorientierte Verwaltung der Programmvariablen, erhöht die Sicherheit gegenüber Programmierfehlern und verbessert die Programmdokumentation.

<u>Def. 6</u>

Eine <u>Assignation M über A</u> ist eine Zeichenkette der Form

<u>assi</u> w

mit $w \in W(A)$.

Es gelte:

$MVar(M) \leftrightharpoons MVar(w)$,
$Var(M) \leftrightharpoons Var(w)$,
$LPA(M) \leftrightharpoons \emptyset$,
$GPA(M) \leftrightharpoons \emptyset$,
$FVVar(M) \leftrightharpoons Var(w)$,
$GVVar(M) \leftrightharpoons \emptyset$,
$VKonst(M) \leftrightharpoons FKonst(w)$.

Strukturgraph von M:

Bemerkung

Wie schon bei der Definition von Termen und Formeln erwähnt wurde, wollen wir unsinnige Programmkonstruktionen möglichst früh, d. h. nach Möglichkeit bereits syntaktisch, von der weiteren Betrachtung ausschließen.
Hierzu dient die Definition der Mengen
MVar(M) (Modifizierbare Variable von M),
Var(M) (Variable von M),
LPA(M) (Namen lokaler Prozeduraufrufe von M),
GPA(M) (Information über globale Prozeduraufrufe von M),
FVVar(M) (Freie Verifikationsvariable von M),
GVVar(M) (Gebundene Verifikationsvariable von M),
VKonst(M) (Verifikationskonstante von M)
bei (partiell-spezifizierten) Moduln M, wobei MVar(M) die Überprüfung der "Disjunktheit" der Zweige eines parallelen Moduls ermöglicht, Var(M) die der "Vollständigkeit" der Vereinbarung von Modulvariablen, LPA(M) eine eineindeutige Zuordnung zwischen lokalen Prozeduraufrufen und lokal-aufrufbaren Moduln sowie eine Überprüfung der Aufrufhierarchie lokal-aufrufbarer Moduln, GPA(M) die Überprüfung der Aufrufhierarchie global-aufrufbarer Moduln sowie die Überprüfung der syntaktischen Konsistenz derjenigen Prozeduraufrufe, die sich auf den gleichen global-aufrufbaren Modul beziehen, und FVVar(M), GVVar(M) und VKonst(M) ermöglichen schließlich eine konsistente Verwendung des der Modul-Verifikation zugeordneten Anwenderalphabets.
Daneben ist es für den weiteren Aufbau unserer Theorie zweckmäßig, Zugriffsmöglichkeiten auf z. B. den in einem Prozeduraufruf M vorkommenden Modulnamen nam(M), auf die Anzahl der Eingangs-Parameter est(M) bzw. Ausgangsparameter ast(M) eines globalen Prozeduraufrufs M oder die in einem Modul vorkommenden Zusicherungen bereits bei der Definition der Syntax zur Verfügung zu stellen.

Zur Verdeutlichung der intendierten semantischen Struktur stellen wir die Moduln bereits bei der Syntaxdefinition informell in dem Modell dar (Strukturgraph), auf das sich später die semantische Analyse beziehen wird.

Def. 7

Ein lokaler Prozeduraufruf M über A ist eine Zeichenkette der Form

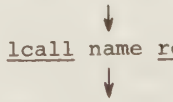
lcall name res w

mit name $\in$ MNam(A) $\wedge$ Var(w) $\subseteq$ Var(A) $\wedge$
$(\exists\ (A',J') \in \mathfrak{A})((A,J) \subseteq (A',J') \wedge w \in W(A'))$.

Es gelte:

MVar(M) $\leftrightharpoons$ MVar(w),
Var(M) $\leftrightharpoons$ Var(w),
LPA(M) $\leftrightharpoons$ {name},
GPA(M) $\leftrightharpoons$ $\emptyset$,
FVVar(M) $\leftrightharpoons$ Var(w),
GVVar(M) $\leftrightharpoons$ $\emptyset$,
VKonst(M) $\leftrightharpoons$ FKonst(w),
nam(M) $\leftrightharpoons$ name (Name von M).

Strukturgraph von M:



<u>Def. 8</u>

Ein <u>globaler Prozeduraufruf M über A</u> ist eine Zeichenkette der Form

$$\underline{\text{gcall}}\ \text{name}((t_1,t_2,\ldots,t_m),(v_1,v_2,\ldots,v_n))\ \underline{\text{res}}\ w$$

mit

$\text{name} \in \text{MNam}(A) \wedge$

$(\forall\ 1 \le i \le m)\, t_i \in \text{Term}(A) \wedge$

$(\forall\ 1 \le i \le n)\, v_i \in \text{Var}(A) \wedge$

$\text{Disj}(\{v_i\}, 1 \le i \le n) \wedge$

$(\exists\ (A',J') \in \mathfrak{A})((A,J) \subseteq (A',J') \wedge w \in W(A')) \wedge$

$\text{MVar}(w) = \{v_i \mid 1 \le i \le n\} \wedge$

$\text{Var}(w) \subseteq \bigcup_{i=1}^{m} \text{Var}(t_i) \cup \text{MVar}(w).$

Es gelte:

$\text{MVar}(M) \leftrightharpoons \{v_i \mid 1 \le i \le n\},$

$\text{Var}(M) \leftrightharpoons \bigcup_{i=1}^{n} \text{Var}(t_i) \cup \text{MVar}(M),$

$\text{LPA}(M) \leftrightharpoons \emptyset,$

$\text{GPA}(M) \leftrightharpoons \{(\text{name}, \sum_{i=1}^{m} \text{wst}(t_i), n)\},$

$\text{FVVar}(M) \leftrightharpoons \text{Var}(w),$

$\text{GVVar}(M) \leftrightharpoons \emptyset,$

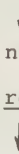
$\text{VKonst}(M) \leftrightharpoons \text{FKonst}(w),$

$\text{nam}(M) \leftrightharpoons \text{name}$

$\text{est}(M) \leftrightharpoons \sum_{i=1}^{n} \text{Wst}(t_i)$ (Eingangsstelligkeit von M),

$\text{ast}(M) \leftrightharpoons n$ (Ausgangsstelligkeit von M).

Strukturgraph von M:



Bemerkung

Um den Unterschied zu Assignationen deutlich hervorzuheben, wird für die Beschreibung der Bedeutung von Prozeduraufrufen in Form von Wertzuweisungen ein zusätzliches Wortsymbol res[1] verwendet. Wie die folgenden Ausführungen noch zeigen werden, ist es auch zweckmäßig, die Bedeutung von parallelen Moduln analog zur Bedeutung von Prozeduraufrufen zu beschreiben. Diese Zweckmäßigkeit entfällt bei unseren anderen Moduln, denn unter der Voraussetzung, daß die Bedeutung von Prozeduraufrufen und parallelen Moduln in Form von Wertzuweisungen dokumentiert ist, läßt sich bei diesen Moduln der Verifikationsprozeß "einfach" beschreiben und handhaben - im Unterschied zu Prozeduraufrufen und parallelen Moduln, wo dies im allgemeinen erst über den Kunstgriff der Beschreibung der Modulbedeutung in Form von Wertzuweisungen ermöglicht wird.
Die Beschränkung von Modulschachtelungen auf Prozeduraufrufe führt zu einem hierarchischen Aufbau der Syntaxdefinition unserer Rechenalgorithmen.

---

1) Resultat des Prozeduraufrufs bzw. des parallelen Moduls.

Def. 9

Ein einfach-serieller Modul M über A ist eine Zeichenkette der Form

$$M_1 M_2 \ldots M_n$$

mit

$n \geq 1 \wedge$

$(\forall\ 1 \leq i \leq n)(M_i$ ist Assignation über A $\vee$

M_i ist lokaler oder globaler Prozeduraufruf über A) $\wedge$

$Disj(LPA(M_i), 1 \leq i \leq n) \wedge$

$Konsi(GPA(M_i), 1 \leq i \leq n)$.

Es gelte:

$(\forall\ E \in \{MVar, Var, LPA, GPA, FVVar, GVVar, VKonst\})$

$$E(M) \leftrightharpoons \bigcup_{i=1}^{n} E(M_i)$$

Strukturgraph von M:

Bemerkung

Neben den aus der Konkatenation von Assignationen und/oder Prozeduraufrufen bestehenden einfach-seriellen Moduln gibt es "komplex"-serielle Moduln, die in Form lokal-aufrufbarer Moduln bzw. als Hauptprogramme global-aufrufbarer Moduln vorkommen.
Die Einfachheit der Modul-Berechnungsstruktur einfach-serieller Moduln zeigt sich im Strukturgraph dadurch, daß nur ein Kantenzug vom Eingang zum Ausgang führt.

Def. 10

Ein paralleler Modul M über A ist eine Zeichenkette der Form

$$\begin{array}{l} \underline{\text{para}}(\text{zahl})\{\Phi\} \\ \quad \underline{\text{p-z}}_1 : M_1 \ \underline{\text{p-exit}} \ \{\Phi_1\} \\ \quad \underline{\text{p-z}}_2 : M_2 \ \underline{\text{p-exit}} \ \{\Phi_2\} \\ \quad \vdots \\ \quad \underline{\text{p-z}}_n : M_n \ \underline{\text{p-exit}} \ \{\Phi_n\} \\ \underline{\text{p-end}}(\text{zahl})\underline{\text{res}}\ w \end{array}$$

mit $n \geq 2 \wedge$

$(\forall\ 1 \leq i \leq n)(M_i$ ist einfach-serieller Modul über A $\wedge$

$z_i \in \text{NNam}(A) \wedge \text{fN}(A)(z_i) = i) \wedge$

$\text{zahl} \in \text{NNam}(A) \wedge \text{fN}(A)(\text{zahl}) = n \wedge$

$\text{Disj}(\text{LPA}(M_i), 1 \leq i \leq n) \wedge \text{Disj}(\text{MVar}(M_i), 1 \leq i \leq n) \wedge$

$\text{Konsi}(\text{GPA}(M_i), 1 \leq i \leq n) \wedge$

$(\exists (A', J') \in \mathfrak{A})((A, J) \subseteq (A', J') \wedge w \in W(A') \wedge$

$(\forall\ 1 \leq i \leq n)\ \Phi, \Phi_i \in \text{Form}(A') \wedge$

$$\bigcup_{i=1}^{n} \text{VKonst}(M_i) \subseteq \bigcup_{i,j \in N_0} \text{FKonst}(A')^{(i,j)}) \wedge$$

$$\bigcup_{i=1}^{n} \text{Var}(M_i) \cap (\bigcup_{i=1}^{n} \text{GVar}(\Phi_i) \cup \text{GVar}(\Phi)) = \emptyset \wedge$$

$$\text{MVar}(w) = \bigcup_{i=1}^{n} \text{MVar}(M_i) \wedge$$

$$\text{Var}(w) \subseteq \bigcup_{i=1}^{n} \text{Var}(M_i).$$

Es gelte:

$$(\forall E \in \{MVar, Var, LPA, GPA\}) \leftrightharpoons E(M) = \bigcup_{i=1}^{n} E(M_i),$$

$$FVVar(M) \leftrightharpoons \bigcup_{i=1}^{n} (FVVar(M_i) \cup FVar(\Phi_i)) \cup FVar(\Phi),$$

$$GVVar(M) \leftrightharpoons \bigcup_{i=1}^{n} (GVVar(M_i) \cup GVar(\Phi_i)) \cup GVar(\Phi),$$

$$VKonst(M) \leftrightharpoons \bigcup_{i=1}^{n} (VKonst(M_i) \cup FKonst(\Phi_i)) \cup FKonst(\Phi) \cup FKonst(w),$$

$\Phi_A(M) \leftrightharpoons \Phi_A(M,i) \leftrightharpoons \Phi$ für $1 \leq i \leq n$

(<u>Anfangszusicherung</u> von M bzw. dem i-ten parallelen Zweig von M),

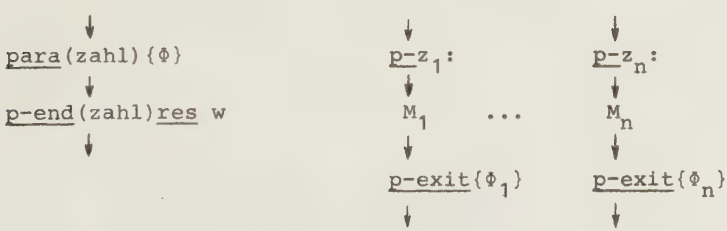
$\Phi_E(M,i) \leftrightharpoons \Phi_i$ für $1 \leq i \leq n$

(<u>Endzusicherung</u> des i-ten parallelen Zweigs von M),

$MVar(M,i) \leftrightharpoons MVar(M_i)$ für $1 \leq i \leq n$

(Menge der modifizierbaren Variablen von M_i).

Strukturgraph von M:



Eine Zeichenkette M_p der Form

$\underline{p}\text{-}z_i$: M_1 $\underline{p}\text{-}exit\{\Phi_i\}$

heißt (i-ter) <u>paralleler Zweig von M</u>,

$\Phi_A(M_p) \leftrightharpoons \Phi_A(M)$ <u>Anfangszusicherung von M_p</u> und

$\Phi_E(M_p) \leftrightharpoons \Phi_E(M,i)$ <u>Endzusicherung vom M_p</u>.

Bemerkung

Das Wortsymbol p- dient nicht nur als (redundante) Anfangsmarkierung paralleler Zweige, sondern erleichtert zusammen mit der (ebenfalls redundanten) Angabe der Anzahl an parallelen Zweigen die Handhabung paralleler Moduln im Rahmen unserer Modellbildung.
Jedem parallelen Zweig M_p ist mittels $\Phi_A(M_p) = \Phi$ eine Anfangszusicherung zugeordnet, d. h. jeder parallele Zweig besitzt eine Anfangs- und Endzusicherug und kann damit bezüglich der Programmverifikation als weitgehend selbständige Einheit gehandhabt werden. Diese Selbständigkeit der parallelen Zweige wird auch durch den Strukturgraph ausgedrückt, der aus n+1 Zusammenhangskomponenten[1] besteht. Nur die "Hauptkomponente" "→ para(zahl){Φ} → p-end(zahl)res w →" des Strukturgraphs wird mit anderen Moduln sequentiell verknüpfbar sein, wobei die Wertzuweisung in res w die Anwendung eines deterministischen Berechnungsbegriffs auf parallele Moduln ermöglicht. Würde anstelle von res w eine Zusicherung $\{\Phi_E\}$ stehen, so wäre damit ein nichtdeterministischer Berechnungsbegriff verknüpft (falls Φ_E zur Definition der Modulberechnung verwendet würde).

---

1) Die verwendeten graphentheoretischen Grundbegriffe stützen sich auf Müller [1975].

Def. 11

Ein dezentral-verzweigender Modul M über A ist eine Zeichenkette der Form

M_1
<u>if</u> B_1 M_2 <u>exit</u>
M_3
<u>if</u> B_2 M_4 <u>exit</u>
⋮
<u>if</u> B_n M_{2n} <u>exit</u>
M_{2n+1}
<u>end</u>

mit $n \geq 1 \wedge$

$(\forall\ 1 \leq i \leq 2n+1 (M_i = \Lambda \vee M_i$ ist einfach-serieller Modul über A) $\wedge$

$\mathrm{Disj}(\mathrm{LPA}(M_i), 1 \leq i \leq n)$[1] $\wedge$

$\mathrm{Konsi}(\mathrm{GPA}(M_i), 1 \leq i \leq n) \wedge$

$(\forall\ 1 \leq i \leq n)(B_i$ ist Gleichung über A).

Es gelte:

$$(\forall\ E \in \{\mathrm{MVar}, \mathrm{LPA}, \mathrm{GPA}, \mathrm{GVVar}\}) E(M) \leftrightharpoons \bigcup_{i=1}^{2n+1} E(M_i),$$

$$\mathrm{Var}(M) \leftrightharpoons \bigcup_{i=1}^{2n+1} \mathrm{Var}(M_i) \cup \bigcup_{i=1}^{n} \mathrm{Var}(B_i),$$

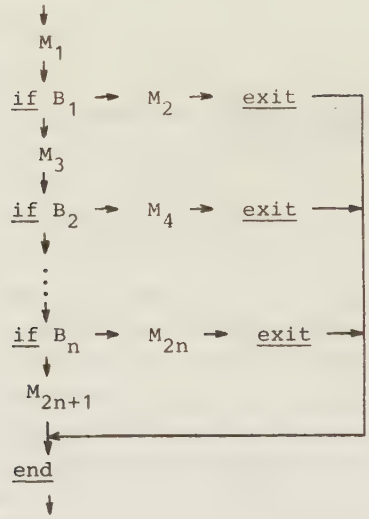
$$\mathrm{FVVar}(M) \leftrightharpoons \mathrm{Var}(M),$$

$$\mathrm{VKonst}(M) \leftrightharpoons \bigcup_{i=1}^{2n+1} \mathrm{VKonst}(M_i) \cup \bigcup_{i=1}^{n} \mathrm{FKonst}(B_i).$$

---

1) Für $M_i = \Lambda$ gelte: $(\forall E \in \{\mathrm{MVar}, \mathrm{Var}, \mathrm{LPA}, \mathrm{GPA}, \mathrm{FVVar}, \mathrm{GVVar}, \mathrm{VKonst}\})$ $E(M_i) \leftrightharpoons \emptyset$.

Strukturgraph von M:



Bemerkung

Bei der Verzweigungsstruktur von Moduln unterscheiden wir zwischen dezentral-verzweigenden und zentral-verzweigenden Moduln, wobei beide Verzweigungsstrukturen auch iteriert in Form von Schleifen auftreten können. Diese beiden Verzweigungsstrukturen ergeben sich u. a. aus der Frage nach zweckmäßigen Normierungen für mehrfach-geschachtelte if-then-else-Anweisungen mit dem Ziel einer klareren und einfacher handhabbaren Darstellungsform.
So entspricht z. B. unserem dezentral verzweigenden Modul die folgende, "else-seitig"-geschachtelte Anweisung:

M_1 if B_1 then M_2
else M_3 if B_2 then M_4
else M_5 if B_3 ...
⋮
fi
fi
fi

Es dürfte dabei direkt ersichtlich sein, daß unsere gewählte Darstellungsform für dezentral-verzweigende Moduln weitaus übersichtlicher ist als obige unter Verwendung geschachtelter if-then-else-Anweisungen.

Def. 12

Ein zentral-verzweigender Modul M über A ist eine Zeichenkette der Form

$$
\begin{array}{l}
\underline{\text{case}}\ \text{mod}\ (B_1, B_2, \ldots, B_n) \\
\quad \text{Marke}_1 : M_1\ \underline{\text{exit}} \\
\quad \text{Marke}_2 : M_2\ \underline{\text{exit}} \\
\quad \vdots \\
\quad \text{Marke}_n : M_n\ \underline{\text{exit}} \\
\quad \underline{\text{otherwise}} : M_{n+1} \\
\underline{\text{end}}
\end{array}
$$

mit $n \geq 2 \wedge \text{mod} \in \{\Lambda, \underline{\text{one}}\} \wedge$

$(\forall\ 1 \leq i \leq n+1)(M_1 = \Lambda \vee M_i$ ist einfach-serieller Modul über A$) \wedge$

$(\forall\ 1 \leq i \leq n)(B_i$ ist Gleichung über A$\ \wedge (\text{Marke}_i = B_i \vee$

$(\exists\ \text{zahl} \in \text{NNam}(A))(\text{fN}(A)(\text{zahl}) = i \wedge$

$\text{Marke}_i = \underline{\text{c}}\text{-zahl})) \wedge$

$\text{Disj}(\{B_i\}, 1 \leq i \leq n) \wedge$

$\text{Disj}(\text{LPA}(M_i), 1 \leq i \leq n+1) \wedge$

$\text{Konsi}(\text{GPA}(M_i), 1 \leq i \leq n+1).$

Es gelte:

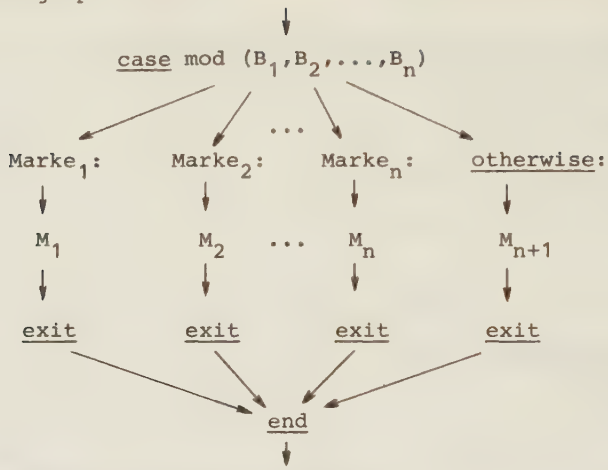
$$(\forall E \in \{\text{MVar}, \text{LPA}, \text{GPA}, \text{GVVar}\})E(M) \leftrightarrows \bigcup_{i=1}^{n+1} E(M_i),$$

$$\text{Var}(M) \leftrightarrows \bigcup_{i=1}^{n+1} \text{Var}(M_i) \cup \bigcup_{i=1}^{n} \text{Var}(B_i),$$

$$\text{FVVar}(M) \leftrightarrows \text{Var}(M),$$

$$\text{VKonst}(M) \leftrightarrows \bigcup_{i=1}^{n+1} \text{VKonst}(M_i) \cup \bigcup_{i=1}^{n} \text{FKonst}(B_i).$$

Strukturgraph von M:



Bemerkung

Im Unterschied zu dezentral-verzweigenden Moduln stehen bei zentral-verzweigenden Moduln alle Bedingungen, die die Berechnungen des Moduls steuern können, an einer zentralen Stelle am Modulanfang. Damit ist eine parallele, gleichzeitige Auswertung der Bedingungen $B_1, B_2, \ldots, B_n$ des zentral-verzweigenden Moduls möglich. Ist für jede zulässige Belegung höchstens eine der Bedingungen B_i $(1 \leq i \leq n)$ erfüllt, so kann die Programmausführungszeit noch zusätzlich dadurch verkürzt werden, daß bereits bei Erfüllung eines B_i verzweigt wird, auch wenn die Auswertung anderer B_j noch nicht abgeschlossen ist. Zur Darstellung dieses Sachverhalts dient das Wortsymbol one.

Können mehrere der Bedingungen B_i bei einer zulässigen Belegung erfüllt sein, so muß - bei Gewährleistung einer deterministischen Programmausführung - vereinbart sein, nach welcher dieser Bedingungen zu verzweigen ist: Es muß also eine Vorschrift existieren, die die gewünschte eindeutige Verzweigung ermöglicht.

Zweckmäßigerweise interpretiert man die Reihenfolge, in der die $B_1, B_2, \ldots, B_n$ aufgelistet sind als eine solche Vorschrift und verzweigt nach Min{i,j,k} wenn z.B. B_i, B_j und B_k erfüllt sind. Obwohl auch in diesem Fall die $B_1, B_2, \ldots, B_n$ parallel ausgewertet werden können, kann bei Erfüllung von B_i erst dann verzweigt werden, wenn die Auswertung aller B_j mit $j<i$ abgeschlossen ist.

Bei zentral-verzweigenden Moduln ist jedem Berechnungszweig eine Marke vorangestellt. Prinzipiell sind derartige Marken überflüssig, denn durch Abzählen der Bedingungen $B_1, B_2, \ldots, B_n$ und der Vorwärtssprünge <u>exit</u> läßt sich die gewünschte Zuordnung zwischen Bedingung und Berechnungszweig ja herstellen - vorausgesetzt wir vereinbaren zwischen den Bedingungen $B_1, B_2, \ldots, B_n$ und den Berechnungszweigen $M_1, M_2, \ldots, M_n$ die naheliegende Beziehung, daß für alle $1 \leq i \leq n$ B_i der Berechnungszweig M_i zugeordnet ist. Mit einem derartigen, nur implizit gegebenen Bezug zwischen den B_i's und den M_i's lassen sich Forderungen aus der Methodik der Programmierung (wie z.B. die nach Lokalität) nur für kleine n(n=2,3,...?) und kurze Zeichenfolgen $B_1, B_2, \ldots, B_n$ sowie $M_1, M_2, \ldots, M_n$ erfüllen, d.h. in vielen Fällen sind für die Kennzeichnung des Bezugs zwischen den B_i's und M_i's Marken zweckmäßig. Da alle B_i's verschieden sind, können die B_i's selbst als Marken verwendet werden. Ausgehend von dieser ausführlichsten Darstellungsform des Bezugs zwischen den B_i's und M_i's können wir dann mithilfe pragmatisch begründeter Konventionen und Abkürzungen zu kürzeren Darstellungsformen kommen, von denen jedoch nur eine mit in unsere weiteren Überlegungen aufgenommen wird:
Anstelle einer Bedingung B_i ist zur Kennzeichnung eines Zweiges auch der B_i zugeordnete Index, d. h. eine natürliche Zahl erlaubt. Das Sonderzeichen <u>c-</u> dient dabei zunächst nur zur Erleichterung der Programm-Lesbarkeit.

Nachdem die Marken vor den einzelnen Berechnungszweigen nicht als überflüssig angesehen werden können, stellt sich noch die Frage, ob auf die Auflistung der Bedingungen $B_1, B_2, \ldots, B_n$ hinter dem Wortsymbol case nicht grundsätzlich verzichtet werden sollte - was dann möglich ist, wenn diese Bedingungen direkt den zugehörigen Berechnungszweigen vorangestellt werden -, um ein evtl. doppeltes Aufschreiben dieser Bedingungen zu ersparen.

Die Antwort ergibt sich von selbst, wenn wir die Eigenschaft der "Lokalität" beider Alternativen untersuchen: Die Zeichenkette case mod$(B_1, B_2, \ldots, B_n)$ stellt auf relativ kleinem Raum - der oft auf einen Blick überschaubar ist - die unterschiedlichen Berechnungsbedingungen dar und ermöglicht eine weitestgehend umgebungsunabhängige Semantikdefinition, was bei der zur Diskussion stehenden Alternative nur unter Verzicht auf eine Parallelauswertung von $B_1, B_2, \ldots, B_n$ der Fall ist. Der Gewinn an "Lokalität" sowie "natürlicher" und leichter Programminterpretation wird also höher bewertet als die Einsparung von Schreibarbeit bei der Programmerstellung, zumal ja durch die alternative Verwendung von natürlichen Zahlen (bzw. zusätzlicher pragmatisch begründeter Konventionen und Abkürzungen) der Aufwand an prinzipiell überflüssiger Schreibarbeit begrenzbar ist.
Ein Verzicht auf Redundanz erscheint höchstens dort vertretbar, wo damit kein derartiger Gewinn erzielbar ist und - wie z. B. mit der Verwendung von "then" als "Sprungmarke" bei "if B then" - die Redundanz im wesentlichen nur zur Erzielung gewohnter Darstellungsformen dient.

Da die Bedingungen $B_1, \ldots, B_n$ Gleichungen der Form $t_1 = t_1'$, $t_2 = t_2'$, $t_n = t_n'$ sind, kann zwischen den Bedingungen $B_1, B_2, \ldots, B_n$ u. a. eine Beziehung bestehen derart, daß gilt $t_1 = t_2 = \ldots = t_n$. Zur Sichtbarmachung dieses Sachverhalts und zur Einsparung von Schreibarbeit bietet sich die folgende Abkürzung für zentralverzweigende Moduln an:

<u>case</u> t_1 <u>of</u> mod $(t'_1, t'_2, \ldots, t'_n)$

Marke_1 : M_1 <u>exit</u>

Marke_2 : M_2 <u>exit</u>

$\vdots$

Marke_n : M_n <u>exit</u>

<u>otherwise</u> : M_{n+1} <u>exit</u>

<u>end</u>

mit $(\forall\ 1 \leq i \leq n)\ (\text{Marke}_i = t'_i \vee (\exists\ \text{zahl} \in \text{NNam}(A))(\text{fN}(A)(\text{zahl}) = i \wedge \text{Marke}_i = \underline{c}\text{-zahl}))$.

Während die Verwendung des Sonderzeichens <u>c-</u> bei unseren zentral-verzweigenden Moduln im wesentlichen nur zur Erleichterung der Programm-Lesbarkeit dient, ist es bei obiger Abkürzung notwendig, da ohne die Verwendung von <u>c-</u> die Marken ihre Funktion als eindeutige Identifikatoren verlieren.

Beispiel:

<u>case</u> x of y^2,1,0

1 : M_1 <u>exit</u> --> $\text{Marke}_1 \in \text{NNam}(A)$

1 : M_2 <u>exit</u> --> $\text{Marke}_2 = t_2$

0 : M_3 <u>exit</u> --> $\text{Marke}_3 = t_3$

<u>otherwise</u> M_4 <u>exit</u>

<u>end</u>

Selbst diese Darstellungsform ist noch immer allgemeiner als die meisten selektiven Anweisungen höherer Programmiersprachen, denn z.B. sind im Unterschied zur case-Anweisung von PASCAL hier beliebige Anwenderterme $t_1', t_2', \ldots, t_n'$ zugelassen. So läßt sich z.B.

```
case x+y of (z,2•z,3•z)
  z   : ... exit
  2•z : ... exit
  3•z : ... exit
  otherwise : ... exit
end
```

nicht direkt als case-Anweisung von PASCAL darstellen[1].

Sind die Terme $t_1', t_2', \ldots, t_n'$ Individuenkonstante, so entspricht unser zentral-verzweigender Modul weitgehend den selektiven Anweisungen (case-Anweisungen) von PASCAL und ALGOL 68. Auch die bedingten Anweisungen höherer Programmiersprachen, wie z.B. die if-then-else-Anweisung von ALGOL 60, lassen sich im Rahmen dieses Sonderfalls darstellen:

```
case x>y of true
     true : ... exit
     otherwise : ... exit
end
```

Ist t zusätzlich eine Individuenvariable, so erhalten wir Verallgemeinerungen des Zielverteilers von ALGOL 60:

```
case x of 1,2,3,4
     1 : ... exit
     2 : ... exit
     3 : ... exit
     4 : ... exit
     otherwise : ... exit
end
```

---

1) Zur Steigerung der Lesbarkeit wurde für Anwenderterme $t, t_1', \ldots, t_n'$ hier die aus der Mathematik vertraute Darstellungsform und nicht unsere Normalform-Darstellung verwendet.

Ein weiterer Spezialfall liegt dann vor, wenn der Term t ein konstanter Term ist:

```
case true of x<y, x=y
     x<y : ... exit
     x=y : ... exit
     otherwise : ... exit
end
```

Das Beispiel zeigt eine Anwendungsmöglichkeit im Sinne einer mathematischen Fallunterscheidung und entspricht einer bedingten FORTRAN-Anweisung, wenn wir x einem arithmetischen FORTRAN-Term und y der Zahl 0 gleichsetzen.

Daneben lassen sich geschachtelte if-then-else-Anweisungen damit übersichtlicher darstellen:

```
case 5 of x,y,z
     x : A1 exit
     y : A2 exit
     z : A3 exit
     otherwise : A4 exit
end
```

ist z. B. übersichtlicher als

```
if x=5 then A1
       else if y=5 then A2
                   else if z=5 then A3
                               else A4
                   fi
            fi
fi
```

Können mehrere der Bedingungen $B_1,\ldots,B_n$ eines zentralverzweigenden Moduls bei einer zulässigen Belegung gleichzeitig erfüllt sein, so wäre auch festlegbar, daß alle die Berechnungszweige M_i parallel ausgeführt werden, deren zugeordnete Bedingungen B_i erfüllt sind. Es wäre dann möglich, daß in Abhängigkeit von der jeweils vorliegenden Belegung Berechnungszweige zu parallel ausführbaren Gruppen zusammengefaßt werden, d.h. die Belegung die Funktion der Steuerung parallel ausführbarer Berechnungszweige übernimmt. Bei Spezialanwendungen könnte diese Interpretationsart für zentral-verzweigende Moduln durchaus elegante Darstellungsformen implizieren - doch die Frage, ob diese Darstellungsform unter den allgemeinen Gesichtspunkten der Methodik der Programmierung verwendet werden sollte, ist damit noch lange nicht positiv beantwortet. Insbesondere die Verifizierung derartiger Moduln dürfte große Schwierigkeiten aufwerfen. Diese dritte Interpretationsart für zentral-verzweigende Moduln könnte auch zur Darstellung paralleler Moduln herangezogen werden, wenn gewährleistet ist, daß für alle zulässigen Belegungen stets jede der Bedingungen $B_1,B_2,\ldots,B_n$ erfüllt ist. Da in diesem Fall jedoch die Auflistung der Bedingungen $B_1,B_2,\ldots,B_n$ und der Marken Marke_1, $\text{Marke}_2,\ldots$ Marke_n überflüssig wird, erscheint eine spezielle Darstellungsform für parallele Moduln gerechtfertigt, obwohl strenggenommen parallele Moduln zur Klasse der zentral-verzweigenden Moduln gehören. Darüber hinaus haben wir vorausgesetzt, daß bei unseren parallelen Moduln Konflikte rein syntaktisch und nicht erst durch Steuerungsmechanismen (z. B. mithilfe von Semaphors) ausgeschlossen werden. Eine syntaktische Möglichkeit zur Vermeidung von Konflikten existiert für die dritte, allgemeinste Interpretationsart zentral-verzweigender Moduln jedoch nicht.

Ein Zusammenhang besteht auch zwischen zentral- und dezentral-verzweigenden Moduln, denn dezentral-verzweigende Moduln mit $M_1=M_3=M_5=\ldots=M_{2n+1}=\Lambda$ entsprechen zentral-verzweigenden Moduln. Man kann somit zentral-verzweigende Moduln als Spezialfälle dezentral-verzweigender Moduln auffassen, wobei pragmatische Überlegungen jedoch eine ausgezeichnete Darstellungsform dieses Spezialfalls rechtfertigen.

Ein weiterer Zusammenhang besteht mit den "Guarded Commands" von Dijkstra [1975], denn die Zeichenfolge "B : M_1 <u>exit</u>" läßt sich auch als "Guarded Command" interpretieren. Alternativ zu deterministischen Interpretationsarten könnten wir analog zu Dijkstra [1975] somit auch noch nichtdeterministische Interpretationsarten für zentral-verzweigende Moduln betrachten. Unser Verzicht auf nichtdeterministische Wertzuweisungen implizierte jedoch den Verzicht auf nichtdeterministische Moduln und damit auch den Verzicht auf eine derartige nichtdeterministische Interpretationsart unserer zentral-verzweigender Moduln.

<u>Kor. 13</u>

Für einfach-serielle, dezentral- und zentral-verzweigende Moduln M gilt:
GVVar(M) = $\emptyset$.

<u>Def. 14</u>

Ein <u>(dezentral-verzweigender until-) Schleifen-Modul M über A</u> ist eine Zeichenkette der Form

<u>loop</u>{Φ}
M_1
<u>if</u> B_1 M_2 <u>exit</u>
M_3
<u>if</u> B_2 M_4 <u>exit</u>
⋮
<u>if</u> B_n M_{2n} <u>exit</u>
M_{2n+1}
<u>repeat</u>
<u>end</u>

mit $n \geq 1 \wedge$

$(\forall\ 1 \leq i \leq 2n+1)(M_i = \Lambda \vee M_i$ ist einfach-serieller Modul über A) $\wedge$

$\mathrm{Disj}(\mathrm{LPA}(M_i), 1 \leq i \leq 2n+1) \wedge$

$\mathrm{Konsi}(\mathrm{GPA}(M_i), 1 \leq i \leq 2n+1) \wedge$

$(\forall\ 1 \leq i \leq n)(B_i$ ist Gleichung über A) $\wedge$

$$(\exists (A',J') \in \mathfrak{A})((A,J) \subseteq (A',J') \wedge \Phi \in \mathrm{Form}(A') \wedge$$

$$\bigcup_{i=1}^{2n+1} \mathrm{VKonst}(M_i) \subseteq \bigcup_{i,j \in N_0} \mathrm{FKonst}^{(i,j)}(A')) \wedge$$

$$\left(\bigcup_{i=1}^{2n+1} \mathrm{Var}(M_i) \cup \bigcup_{i=1}^{n} \mathrm{Var}(B_i)\right) \cap \mathrm{GVVar}(\Phi) = \emptyset.$$

Es gelte:

$$(\forall\ E \in \{\mathrm{MVar}, \mathrm{LPA}, \mathrm{GPA}\})\ E(M) \leftrightharpoons \bigcup_{i=1}^{2n+1} E(M_i),$$

$$\mathrm{Var}(M) \leftrightharpoons \bigcup_{i=1}^{2n+1} \mathrm{Var}(M_i) \cup \bigcup_{i=1}^{n} \mathrm{Var}(B_i),$$

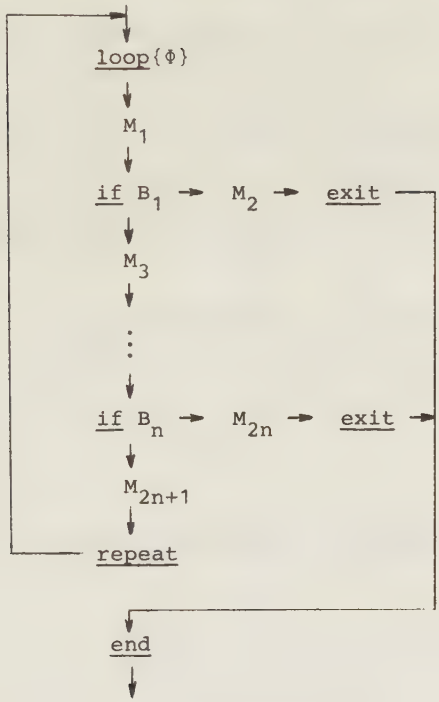
$$\mathrm{FVVar}(M) \leftrightharpoons \mathrm{Var}(M) \cup \mathrm{FVar}(\Phi),$$

$$\mathrm{GVVar}(M) \leftrightharpoons \mathrm{GVar}(\Phi),$$

$$\mathrm{VKonst}(M) \leftrightharpoons \bigcup_{i=1}^{2n+1} \mathrm{VKonst}(M_i) \cup \bigcup_{i=1}^{n} \mathrm{FKonst}(B_i) \cup \mathrm{FKonst}(\Phi).$$

$\Phi_A(M) \leftrightharpoons \Phi$ (<u>Schleifenzusicherung</u> von M).

Strukturgraph von M:



Der <u>Schleifenkörper M_K</u> von M sei die Zeichenkette

$\{\Phi\}$

M_1

<u>if</u> B_2 M_2 <u>exit</u>

⋮

<u>if</u> B_n M_{2n} <u>exit</u>

M_{2n+1}

$\{\Phi\}$

<u>end</u>

$\{T\}$<u>fini</u>

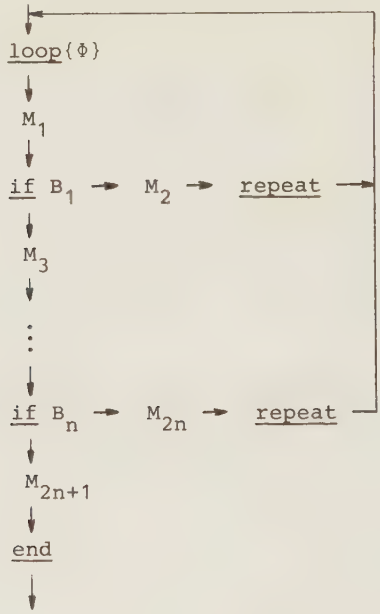
Dabei gelte: $\Phi_A(M_K) = \Phi$,

$\Phi_E(M_K) = T$.

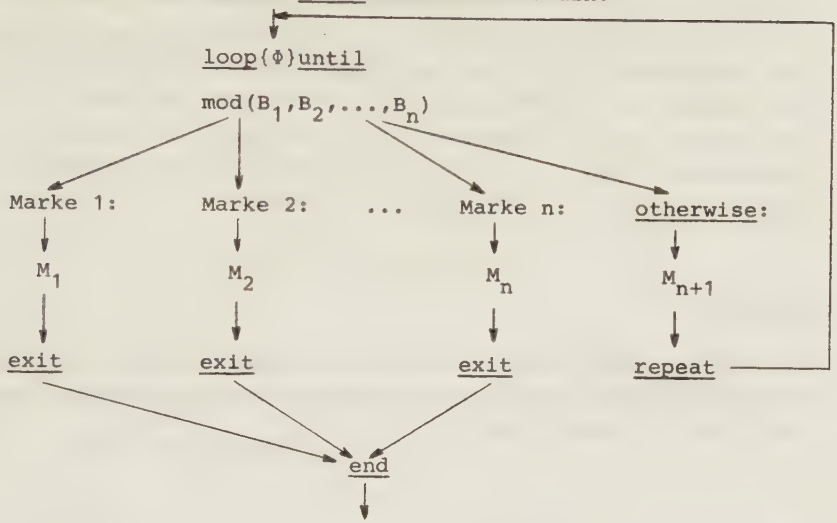
Bemerkung

Mit Rücksicht auf den Umfang dieser Arbeit wollen wir darauf verzichten, alle im Rahmen unserer Theoriebildung ausgezeichneten Klassen von Schleifen-Moduln systematisch zu bearbeiten und uns exemplarisch auf dezentral-verzweigende until-Schleifen-Moduln der angegebenen Form beschränken. Anstelle der Klasse der dezentral-verzweigenden until-Schleifen-Moduln hätten wir auch eine der folgenden drei ausgezeichneten Klassen betrachten können:

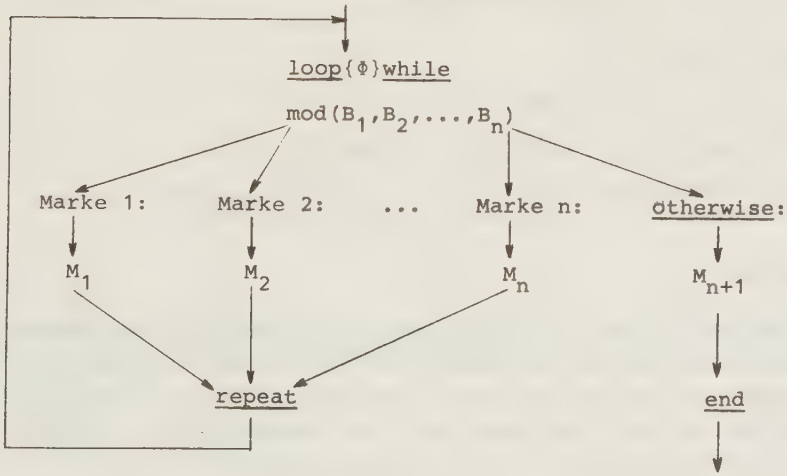
1) dezentral-verzweigende while-Schleifen-Moduln:



2) zentral-verzweigende <u>until</u>-Schleifen-Moduln:



3) zentral-verzweigende <u>while</u>-Schleifen-Moduln[1]:



---

1) Die nichtdeterministische Interpretationsart derartiger Moduln wird in Dijkstra [1975] behandelt.

Die Auszeichnung von Schleifenkörpern erfolgt im Hinblick auf die Programmverifikation: es ist zweckmäßig, den Nachweis der Korrektheit von Schleifen-Moduln auf den Nachweis der Korrektheit des zugehörigen Schleifen-Körpers abzustützen. Die zusätzlichen Wortsymbole until bzw. while bei zentral-verzweigenden Schleifen-Moduln sind redundant und dienen zur Verdeutlichung der jeweiligen Modulklasse.

Def. 15

Ein nackter Modul M über A ist ein einfach-serieller, paralleler, dezentral-verzweigender oder zentral-verzweigender Modul oder ein Schleifen-Modul über A.

Def. 16

Seien $M_1, M_2, \ldots, M_n$ nackte Moduln.

$\mathrm{Konsi}(M_i, 1 \leq i \leq n) \leftrightarrows$

$\mathrm{Disj}(\mathrm{LPA}(M_i), 1 \leq i \leq n) \wedge \mathrm{Konsi}(\mathrm{GPA}(M_i), 1 \leq i \leq n) \wedge$

$(\forall\ 1 \leq i, j \leq n)(i \neq j \rightarrow \mathrm{FVVar}(M_i) \cap \mathrm{GVVar}(M_j) = \emptyset)$

heißt Konsistenzbedingung nackter Moduln $M_1, \ldots, M_n$.

Bemerkung

Während die beiden ersten Konjunktionsglieder der Konsistenzbedingung nackter Moduln einer korrekten Definition von Modulberechnungen dienen, ermöglicht das dritte Konjunktionsglied eine für die Modul-Verifikation klare Unterscheidung zwischen freien und gebundenen Verifikationsvariablen.

Def. 17

Ein kommentierter Modul M über A ist eine Zeichenkette der Form

head k
$M_1 M_2 \ldots M_n$

mit $k \in K^*$ $\wedge$

$(\forall\ 1 \leq i \leq n)(M_i$ ist nackter Modul über A) $\wedge$

Konsi$(M_i, 1 \leq i \leq n)$.

Es gelte:

$(\forall\ E \in \{MVar, Var, LPA, GPA, FVVar, GVVar, VKonst\})\, E(M) \leftrightharpoons \bigcup_{i=1}^{n} E(M_i)$.

"head k" heißt Modulkopf von M.

Strukturgraph von M:

Def. 18

Ein funktions-spezifizierter Modul M über A ist eine Zeichenkette der Form

head k
$\{\Phi\} M_1 M_2 \ldots M_n \{\psi\}$

mit $k \in K^*$ $\wedge$

$(\forall\ 1 \leq i \leq n)(M_i$ ist nackter Modul über A) $\wedge$

Konsi$(M_i, 1 \leq i \leq n)$ $\wedge$

$$(\exists\ (A',J') \in \mathfrak{A})(A,J) \subseteq (A',J') \wedge \Phi,\psi \in \mathrm{Form}(A') \wedge$$

$$\bigcup_{i=1}^{n} \mathrm{VKonst}(M_i) \subseteq \bigcup_{i,j \in N_0} \mathrm{FKonst}^{(i,j)}(A') \wedge$$

$$\bigcup_{i=1}^{n} (\mathrm{FVVar}(M_i) \cup \mathrm{GVVar}(M_i)) \subseteq \mathrm{Var}(A')) \wedge$$

$$\bigcup_{i=1}^{n} \mathrm{Var}(M_i) \cap (\mathrm{GVVar}(\Phi) \cup \mathrm{GVVar}(\psi)) = \emptyset.$$

Es gelte:

$$(\forall\ E \in \{\mathrm{MVar}, \mathrm{Var}, \mathrm{LPA}, \mathrm{GPA}\}\ E(M) \leftrightharpoons \bigcup_{i=1}^{n} E(M_i),$$

$$\mathrm{FVVar}(M) \leftrightharpoons \bigcup_{i=1}^{n} \mathrm{FVVar}(M_i) \cup \mathrm{FVar}(\Phi) \cup \mathrm{FVar}(\psi),$$

$$\mathrm{GVVar}(M) \leftrightharpoons \bigcup_{i=1}^{n} \mathrm{GVVar}(M_i) \cup \mathrm{GVar}(\Phi) \cup \mathrm{GVar}(\psi),$$

$$\mathrm{VKonst}(M) \leftrightharpoons \bigcup_{i=1}^{n} \mathrm{VKonst}(M_i) \cup \mathrm{FKonst}(\Phi) \cup \mathrm{FKonst}(\psi),$$

$$\Phi_A(M) \leftrightharpoons \Phi,$$

$$\Phi_E(M) \leftrightharpoons \psi.$$

"<u>head</u> k" heißt <u>Modulkopf von M</u>.

Strukturgraph von M:

Def. 19

Seien $M_1, M_2, \ldots, M_n$ nackte, kommentierte oder funktions-spezifizierte Moduln über A.

$\mathrm{Konsi}(M_i, 1 \leq i \leq n) \leftrightarrows$

$\mathrm{Disj}(\mathrm{LPA}(M_i), 1 \leq i \leq n) \wedge \mathrm{Konsi}(\mathrm{GPA}(M_i), 1 \leq i \leq n) \wedge$

$(\forall\ 1 \leq i, j \leq n)(i \neq j \rightarrow \mathrm{FVVar}(M_i) \cap \mathrm{GVVar}(M_j) = \emptyset)$

heißt <u>Konsistenzbedingung nicht-aufrufbarer Moduln $M_1, \ldots, M_n$.</u>

Def. 20

Ein <u>lokal-aufrufbarer Modul M über A</u> ist eine Zeichenkette der Form

<u>lproc</u> name
<u>head</u> k
$\{\Phi\}\ M_1 M_2 \ldots M_n\ \{\psi\}$ <u>fini</u>

mit

$\mathrm{name} \in \mathrm{MNam}(A) \wedge$

$k \in K^* \wedge$

$(\forall\ 1 \leq i \leq n)(M_i$ ist nackter oder kommentierter oder funktions-spezifizierter Modul über A$) \wedge$

$\mathrm{Konsi}(M_i), 1 \leq i \leq n) \wedge$

$(\exists\ (A', J') \in \mathfrak{A})((A, J) \underline{c}\ (A', J') \wedge \Phi, \psi \in \mathrm{Form}(A') \wedge$

$$\bigcup_{i=1}^{n} \mathrm{VKonst}(M_i) \underline{c} \bigcup_{i,j \in N_0} \mathrm{FKonst}^{(i,j)}(A') \wedge$$

$$\bigcup_{i=1}^{n} (\mathrm{FVVar}(M_i) \cup \mathrm{GVVar}(M_i)) \underline{c}\ \mathrm{Var}(A')) \wedge$$

$$\bigcup_{i=1}^{n} \mathrm{Var}(M_i) \cap (\mathrm{GVar}(\Phi) \cup \mathrm{GVar}(\psi)) = \emptyset.$$

Es gelte:

$$(\forall\ E \in \{MVar,\ Var,\ LPA,\ GPA\})\ E(M) \leftrightharpoons \bigcup_{i=1}^{n} E(M_i),$$

$$FVVar(M) \leftrightharpoons \bigcup_{i=1}^{n} FVVar(M_i) \cup FVar(\Phi) \cup FVar(\psi),$$

$$GVVar(M) \leftrightharpoons \bigcup_{i=1}^{n} GVVar(M_i) \cup GVar(\Phi) \cup GVar(\psi),$$

$$VKonst(M) \leftrightharpoons \bigcup_{i=1}^{n} VKonst(M_i) \cup FKonst(\Phi) \cup FKonst(\psi),$$

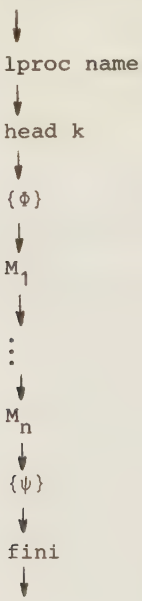
$nam(M) \leftrightharpoons name$

$\Phi_A(M) \leftrightharpoons \Phi$

$\Phi_E(M) \leftrightharpoons \psi.$

"lproc name <u>head</u> k" heißt <u>Modulkopf von M</u>.

Strukturgraph von M:



<u>Bemerkung</u>

Lokal-aufrufbare Moduln sind keine Prozeduren, die von beliebigen Programm-Punkten aus aufrufbar sind, sondern Programmstücke, die fest genau einem Aufrufpunkt (lokalen Prozeduraufruf) zugeordnet sind.

Def. 21

Ein partiell-spezifizierter Modul über A ist ein lokal aufrufbarer, ein funktions-spezifizierter, ein kommentierter oder ein nackter Modul über A.

Def. 22

Die Menge SLMod(A)[1] lokaler (halbgeordneter) Unterprogramm-Systeme über A
sowie die Mengen MVar($\mathfrak{M}$), Var($\mathfrak{M}$), LPA($\mathfrak{M}$), GPA($\mathfrak{M}$), FVVar($\mathfrak{M}$), GVVar($\mathfrak{M}$) und VKonst($\mathfrak{M}$) eines lokalen Unterprogramm-Systems $\mathfrak{M} \in$ SLMod(A) sind definiert durch:

i) Für $\mathfrak{M} \leftrightharpoons \{M\}$, wobei
M ein beliebiger lokal-aufrufbarer Modul über A mit
LPA(M) $= \emptyset$,
gilt: $\mathfrak{M} \in$ SLMod(A)
und
$(\forall\ E \in \{\text{MVar, Var, LPA, GPA, FVVar, GVVar, VKonst}\})$
$E(\mathfrak{M}) \leftrightharpoons E(M)$.

ii) Für $\mathfrak{M} \leftrightharpoons \mathfrak{M}' \cup \{M\}$, wobei
$\mathfrak{M}' \in$ SLMod(A) und
M ein beliebiger lokal-aufrufbarer Modul über A mit
$(\forall\ M' \in \mathfrak{M}')\ \text{nam}(M') \neq \text{nam}(M)\ \wedge$
$\text{nam}(M) \notin \text{LPA}(\mathfrak{M}')\ \wedge$
$(\forall\ \text{name} \in \text{LPA}(M))(\exists\ M' \in \mathfrak{M}')(\text{nam}(M') = \text{name}\ \wedge$
$\text{Var}(\text{name}) = \text{Var}(M')\ \wedge\ \text{MVar}(\text{name}) = \text{MVar}(M'))\ \wedge$
(Konsistenzbedingungen:)
$\text{FVVar}(\mathfrak{M}') \cap \text{GVVar}(M) = \emptyset\ \wedge$
$\text{FVVar}(M) \cap \text{GVVar}(\mathfrak{M}') = \emptyset\ \wedge$
$\text{Konsi}(\text{GPA}(M), \text{GPA}(\mathfrak{M}'))$,
gilt:
$\mathfrak{M} \in$ SLMod(A)
und
$(\forall\ E \in \{\text{MVar, Var, LPA, GPA, FVVar, GVVar, VKonst}\})$
$E(\mathfrak{M}) \leftrightharpoons E(\mathfrak{M}') \cup E(M)$.

---

1) Systeme lokaler Moduln über Anwenderbereich A.

Bemerkung

Vollständig spezifizierte Moduln werden u. a. aus lokalen Unterprogramm-Systemen aufgebaut. Eine notwendige Bedingung für "intellektuell leicht handhabbare" vollständig spezifizierte Moduln ist somit die Beschränkung auf leicht handhabbare lokale Unterprogramm-Systeme.

Def. 23

Sei $\mathfrak{M} \in \mathrm{SLMod}(A)$ beliebig.

a) $\leq'_{\mathfrak{M}} \leftrightharpoons \{(M_1,M_2) \mid M_1,M_2 \in \mathfrak{M} \wedge (M_1 \neq M_2 \rightarrow \mathrm{nam}(M_1) \in \mathrm{LPA}(M_2))\}$

b) $\leq_{\mathfrak{M}} \leftrightharpoons \bigcup_{i \in N_0} (\leq'_{\mathfrak{M}})^i$ (transitive Hülle von $\leq'$) mit $(\leq')^0 \leftrightharpoons \{(M,M) \mid M \in \mathfrak{M}\}$.

Kor. 24

Sei $\mathfrak{M} \in \mathrm{SLMod}(A)$ beliebig.
$\leq_{\mathfrak{M}}$ ist Halbordnung auf $\mathfrak{M}$.

Def. 25

a) $\mathfrak{M} \in \mathrm{SLMod}(A)$ heißt <u>vollständiges lokales Unterprogramm-System über A</u>, wenn $(\mathfrak{M}, \leq_{\mathfrak{M}})$ ein größtes Element besitzt.

b) Sei $\mathfrak{M} = \{M_1,M_2,\ldots,M_n\}$ ein vollständiges lokales Unterprogramm-System über A und M_1 das größte Element von $(\mathfrak{M}, \leq_{\mathfrak{M}})$.
M_1 heißt <u>Hauptprogramm von $\mathfrak{M}$</u> und
$M_2,M_3,\ldots,M_n$ heißen <u>lokale Unterprogramme von $\mathfrak{M}$</u>.

Def. 26

Eine Variableninterpretation I über A ist eine Zeichenkette der Form

$$\underline{\text{type}}\ b_I\colon v_1, v_2, \ldots, v_n$$

mit

$b_I \in BKonst(A) \wedge$
$(\forall\ 1 \leq i \leq n)\ v_i \in Var(A) \wedge\ Disj(\{v_i\}, 1 \leq i \leq n)$.

Es gelte: $Var(I) \leftrightharpoons \{v_1, v_2, \ldots, v_n\}$, $b(I) \leftrightharpoons b_I$.

IVar(A) bezeichne die Menge aller Variableninterpretationen über A.

Bemerkung

Unsere Variableninterpretation entspricht der Typ-Vereinbarung höherer Programmiersprachen wie z. B. ALGOL 60.
Da hierbei keine neue Objektklasse (Typ) definiert wird, sondern vielmehr Programmvariablen eine Bereichskonstante und damit eine Objektklasse zugeordnet wird, ist die Bezeichnung "Variableninterpretation" treffender als die Bezeichnung "Typ-Vereinbarung".

Def. 27

Ein vollständiger Modulkopf über $\mathfrak{A}$ ist eine Zeichenkette MK der Form

b-dom	D
gproc	name
head	k
input	$(e_1, e_2, \ldots, e_m)$
output	$(a_1, a_2, \ldots, a_n)$
global	$IG_1, IG_2, \ldots, IG_g$
local	$IL_1, IL_2, \ldots, IL_l$
ref	$name_1, name_2, \ldots, name_r$
v-dom	D'
bounded	$IB_1, IB_2, \ldots, IB_b$

mit folgender Eigenschaft:

1) $D \in BNam(\mathfrak{A})$, wobei o. B. D. A. D der Anwenderbereich (A,J) zugeordnet sei $\wedge$

2) $name \in MNam(A)$ $\wedge$

3) $k \in K^*$ $\wedge$

4) $m \geq 1 \wedge (\forall\ 1 \leq i < m) e_i \in Var(A) \wedge Disj(\{e_i\}, 1 \leq i \leq m)$ $\wedge$

5) $n \geq 1 \wedge (\forall\ 1 \leq i \leq n) a_i \in Var(A) \wedge Disj(\{a_i\}, 1 \leq i \leq n)$ $\wedge$

6) $g \geq 1 \wedge (\forall\ 1 \leq i \leq g) IG_i \in IVar(A) \wedge Disj(Var(IG_i), 1 \leq i \leq g) \wedge$
$\wedge\ Disj(\{b(IG_i)\}, 1 \leq i \leq g)$ $\wedge$

$$\wedge \bigcup_{i=1}^{g} Var(IG_i) = \{e_i \mid 1 \leq i \leq m\} \cup \{a_i \mid 1 \leq i \leq n\} \wedge$$

7) $l \geq 0 \wedge (\forall\ 1 \leq i \leq l) IL_i \in IVar(A) \wedge \bigcup_{i=1} Var(IG_i) \cap \bigcup_{i=1} Var(IL_i) = \emptyset$
$\wedge\ Disj(Var(IL_i), 1 \leq i \leq l) \wedge Disj(\{b(IL_i)\}, 1 \leq i \leq g)$ $\wedge$

8) $r \geq 0 \wedge (\forall\ 1 \leq i \leq r) name_i \in MNam(A) \wedge Disj(\{name_i\}, 1 \leq i \leq r)$ $\wedge$

9) $D' \in BNam(\mathfrak{A})$, wobei o.B.d.A. D' der Anwenderbereich (A',J') zugeordnet sei mit $(A,J) \subseteq (A',J')$ $\wedge$

10) $b \geq 0 \wedge (\forall\ 1 \leq i \leq b) IB_i \in IVar(A') \wedge Disj(Var(IB_i), 1 \leq i \leq b) \wedge$
$\wedge\ Disj(\{b(IB_i)\}, 1 \leq i \leq b)$ $\wedge$

$$\wedge\ \left(\bigcup_{i=1}^{g} Var(IG_i) \cup \bigcup_{i=1}^{l} Var(IL_i)\right) \cap \bigcup_{i=1}^{b} Var(IB_i) = \emptyset$$

Es gelte:

$Dom(MK) \leftrightharpoons (A,J)$ (<u>Berechnungsbereich</u> von MK),
$VDom(MK) \leftrightharpoons (A',J')$ (<u>Verifikationsbereich</u> von MK),
$nam(MK) \leftrightharpoons name$ (<u>Name</u> von MK),
$input(MK) \leftrightharpoons (e_1, e_2, \ldots, e_m)$ (<u>Eingabe-Parameterliste</u> von MK),
$output(MK) \leftrightharpoons (a_1, a_2, \ldots, a_n)$ (<u>Ausgabe-Parameterliste</u> von MK),
$est(MK) \leftrightharpoons m$ (<u>Eingangsstelligkeit</u> von MK),
$ast(MK) \leftrightharpoons n$ (<u>Ausgangsstelligkeit</u> von MK),

$LVar(MK) \leftrightharpoons \bigcup_{i=1}^{l} Var(IL_i)$ (<u>Lokale Variable</u> von MK),

$GVar(MK) \leftrightharpoons \{e_i \mid 1 \leq i \leq m\} \cup \{a_i \mid 1 \leq i \leq n\}$ (<u>Globale Variable</u> von MK),

$NGPA(MK) \leftrightharpoons \{name_i \mid 1 \leq i \leq r\}$ (<u>Namen globaler Prozeduraufrufe</u> von MK),

$BVar(MK) \leftrightharpoons \bigcup_{i=1}^{b} Var(IB_i)$ (<u>Gebundene Variable</u> von MK).

$$b(v,MK) \leftrightharpoons \iota b' \; (b' \in BKonst(A') \wedge$$
$$((\exists 1 \leq i \leq g)(b'=b(IG_i) \wedge v \in Var(IG_i)) \vee$$
$$(\exists 1 \leq i \leq l)(b'=b(IL_i) \wedge v \in Var(IL_i)) \vee$$
$$(\exists 1 \leq i \leq b)(b'=b(IB_i) \wedge v \in Var(IB_i)))$$

(<u>Bereich von v</u>) für $v \in GVar(MK) \cup LVar(MK) \cup BVar(MK)$ beliebig.

<u>Def. 28</u>

Ein <u>vollständig spezifizierter Modul M über $\mathfrak{A}$</u> ist eine Zeichenkette der Form

MK
$\{\Phi\}M_1M_2\ldots M_s\{\psi\}$<u>fini</u>
$U_{11}U_{12}\ldots U_{1n_1}$,
$U_{21}U_{22}\ldots U_{2n_2}$,
$\vdots$
$U_{u1}U_{u2}\ldots U_{un_u}$

mit folgender Eigenschaft:

1) MK ist vollständiger Modulkopf über $\mathfrak{A}$, wobei o.B.d.A.
   $(A,J) = Dom(MK)$ und
   $(A',J') = VDom(MK)$.

2) $\Phi,\psi \in Form(A') \wedge$

3) $(\forall\ 1 \leq i \leq s)$($M_i$ ist nackter, kommentierter oder funktionsspezifizierter Modul über A) $\wedge$

4) $(\forall\ 1 \leq i \leq u)\ \{U_{i1},U_{i2},\ldots,U_{in_i}\} \in SLMod(A) \wedge$

5) Konsistenzbedingungen:

a) $(\forall\ \text{name} \in \bigcup_{i=1}^{s} LPA(M_i))(\exists\ 1 \leq i \leq u)(nam(U_{i1}) = \text{name} \wedge$

$\wedge\ U_{i1}$ ist Hauptprogramm von $\{U_{i1},\ldots,U_{in_i}\} \wedge$

$\wedge\ Var(\text{name}) = Var(U_{i1}) \wedge MVar(\text{name}) = MVar(U_{i1})) \wedge$

b) $\forall\ x\ (x \in NGPA(MK) \leftrightarrow ((\exists\ 1 \leq i \leq s)(\exists\ \text{tripel} \in GPA(M_i))$

$x = P_1^3(\text{tripel}) \vee$

$(\exists\ 1 \leq i \leq u)(\exists\ \text{tripel} \in GPA(\{U_{i1},\ldots,U_{in_i}\})))$

$x = P_1^3(\text{tripel}))) \wedge$

c) $Konsi(M_i, 1 \leq i \leq s) \wedge Konsi(GPA(\{U_{i1},\ldots,U_{in_i}\}), 1 \leq i \leq u) \wedge$

$(\forall\ 1 \leq i \leq s)(\forall\ 1 \leq j \leq u)\ Konsi(GPA(M_i), GPA(\{U_{i1},\ldots,U_{in_i}\})) \wedge$

$Disj(LPA(\{U_{i1},\ldots,U_{in_i}\}), 1 \leq i \leq u) \wedge$

$(\forall\ 1 \leq i \leq s)(\forall\ 1 \leq j \leq u)\ Disj(LPA(M_i), LPA(\{U_{j1},\ldots,U_{jn_j}\}) \wedge$

d) $LVar(MK) = \bigcup_{i=1}^{s} Var(M_i) \cup \bigcup_{i=1}^{u} Var(\{U_{i1},\ldots,U_{in_i}\})) \setminus GVar(MK) \wedge$

e) $GVar(MK) \cup LVar(MK) = \bigcup_{i=1}^{s} FVVar(M_i) \cup \bigcup_{i=1}^{u} FVVar(\{U_{i1},\ldots,U_{in_i}\})$

$\cup\ FVar(\Phi) \cup FVar(\psi) \wedge$

f) $BVar(MK) = \bigcup_{i=1}^{s} GVVar(M_i) \cup \bigcup_{i=1}^{u} GVVar(\{U_{i1},\ldots,U_{in_i}\})$

$\cup\ GVar(\Phi) \cup GVar(\psi) \wedge$

g) $\bigcup_{i=1}^{s} VKonst(M_i) \cup \bigcup_{i=1}^{u} VKonst(\{U_{i1},\ldots,U_{in_i}\}) \subseteq \bigcup_{i,j \in N_0} FKonst^{(i,j)}(A')$.

h) $\forall x\ (x \in \bigcup_{i=1}^{s} MVar(M_i) \cup \bigcup_{i=1}^{n} MVar(\{U_{i1},\ldots,U_{in_i}\})$

$\rightarrow$ x ist nicht Komponente von input(MK))

Strukturgraph von M:

Es gelte:

$Dom(M) \leftrightharpoons DOM(MK)$ (Berechnungsbereich von M),

$VDom(M) \leftrightharpoons VDom(MK)$ (Verifikationsbereich von M),

$nam(M) \leftrightharpoons nam(MK)$ (Name von M),

$input(M) \leftrightharpoons input(MK)$ (Eingabe-Parameterliste von M),

$output(M) \leftrightharpoons output(MK)$ (Ausgabe-Parameterliste von M),

$est(M) \leftrightharpoons est(MK)$ (Eingangsstelligkeit von M),

$ast(M) \leftrightharpoons ast(MK)$ (Ausgangsstelligkeit von M),

$GPA(M) \leftrightharpoons \bigcup_{i=1}^{s} GPA(M_i) \cup \bigcup_{i=1}^{u} GPA(\{U_{i1},\ldots,U_{in_i}\})$ (Information über globale Prozeduraufrufe von M),

$\Phi_A(M) \leftrightharpoons \Phi$ (Anfangszusicherung von M),

$\Phi_E(M) \leftrightharpoons \psi$ (Endzusicherung von M).

$M' \leftrightharpoons MK\{\Phi\}M_1\ldots M_s\{\psi\}$<u>fini</u> heißt <u>Hauptprogramm von M</u>,

wobei $\Phi_A(M') \leftrightharpoons \Phi$ und $\Phi_E(M') \leftrightharpoons \psi$.

U_{ij} mit $1 \leq i \leq u, 1 \leq j \leq n_i$ heißt <u>lokales Unterprogramm von M</u>.

In Analogie zu lokal-aufrufbaren Moduln heißt M auch <u>global-aufrufbarer Modul</u>.
MK heißt auch <u>Modulkopf von M</u>.

$$J_{Var}(M) \leftrightharpoons \left\langle v \mapsto \begin{cases} b(v,MK), & \text{falls } v \in GVar(MK) \cup LVar(MK) \\ \Omega^{1)} & \text{sonst} \end{cases} \middle| v \in Var(A) \right\rangle$$

heißt <u>Berechnungs-Variablen-Interpretation von M</u>.

$$J'_{Var}(M) = \left\langle v \mapsto \begin{cases} b(v,MK), & \text{falls } v \in GVar(MK) \cup LVar(MK) \cup BVar(MK) \\ \Omega & \text{sonst} \end{cases} \middle| v \in Var(A') \right\rangle$$

heißt <u>Verifikations-Variablen-Interpretation von M</u>.

$P_1^2(Dom(M))$ heißt <u>Berechnungsalphabet von M</u>,

$P_1^2(VDom(M))$ heißt <u>Verifikationsalphabet von M</u>.

$(P_2^2(Dom(M)), J_{Var}(M))$ heißt <u>vollständige Interpretation des Berechnungsalphabets von M</u>,

$(P_2^2(VDom(M)), J'_{Var}(M))$ heißt <u>vollständige Interpretation des Verifikationsalphabetes von M</u>,

$GVar(M) \leftrightharpoons GVar(MK)$ (Globale Variable von M),

$LVar(M) \leftrightharpoons LVar(MK)$ (Lokale Variable von M),

$BVar(M) \leftrightharpoons BVar(MK)$ (Gebundene Variable von M).

$b(v,M) \leftrightharpoons b(v,MK)$ heißt Bereich von v für $v \in GVar(M) \cup LVar(M) \cup BVar(M)$ beliebig.

---

1) Für alle Anwenderbereiche (A,J_K) gelte:
$\Omega \in BKonst(A) \wedge J_K(\Omega) = \{\Omega\}$.
(Ω dient sowohl zur Kennzeichnung undefinierter Bereichskonstanten als auch zur Kennzeichnung undefinierter Variablenwerte.)

Bemerkung

Die Konsistenzbedingungen unter 5) dienen unserem Prinzip, unsinnige bzw. "schlampige" Programme möglichst frühzeitig von einer weiteren Bearbeitung auszuschließen:
Nach Bedingung 5a) gibt es zu jedem lokalen Prozeduraufruf im Hauptprogramm ein Unterprogrammsystem, dessen Hauptprogramm das aufgerufene Unterprogramm ist; die im Unterprogramm-System verwendeten Variablen (bzw. die modifizierbaren Variablen) müssen dabei durch die Wertzuweisung des Prozeduraufrufs korrekt spezifiziert sein.
Bedingung 5b) garantiert, daß im Modulkopf unter ref genau die Namen derjenigen globalen Prozeduren (globalen Unterprogramme von M) stehen, die auch im Modul mittels eines globalen Prozeduraufrufs aufgerufen werden.
Bedingung 5c) garantiert die konsistente Verwendung lokaler und globaler Prozeduraufrufe.
Bedingung 5d) bis 5f) garantiert eine konsistente Verwendung der Variablen (lokale, globale, gebundene) und mit Bedingung 5g) eine konsistente Verwendung des Verifikationsalphabets.
Bedingung 5h) erlaubt in der Endzusicherung von M einen normierten Bezug auf die Anfangswerte von Modulberechnungen mittels der Eingabe-Parameter $e_1, e_2, \ldots, e_m$.

Die Disjunktheit der Namen lokal-aufrufbarer Moduln zweier verschiedener Unterprogramm-Systeme ermöglicht im Prinzip eine beliebige räumliche Zusammenstellung der lokal-aufrufbaren Moduln von Unterprogramm-Systemen(z. B. auf verschiedenen Dateien oder in verschiedenen Lochkarten-Stapeln), obwohl natürlich eine Anordnung entsprechend der Ordnung $\leq_{\mathfrak{M}}$ des Unterprogramm-Systems $\mathfrak{M}$ für eine technologische Realisierung (z. B. Übersetzung, Implementierung auf Universalrechner) vorteilhaft sein kann.
Der Bezug auf den durch einen Prozeduraufruf M aufgerufenen Modul M' (wobei nam(M) = nam(M')ist) erfolgt im Rahmen unserer Theoriebildung mithilfe von nam(M') und es ist zweckmäßig, auf Eigenschaften von M' auch über nam(M) zugreifen zu können. Zur Vereinfachung der Darstellungsweise schreiben wir deshalb z.B. Dom(nam(M')) anstelle von Dom(M').

Def. 29

Die Menge SGMod($\mathfrak{A}$) (globaler, halbgeordneter Unter-) Programm-Systeme über $\mathfrak{A}$ ist definiert durch:

i) Für jeden vollständig spezifizierten Modul M über $\mathfrak{A}$ mit GPA(M) = $\emptyset$ gilt {M} $\in$ SGMod($\mathfrak{A}$).

ii) Für $\mathcal{P} = \mathcal{P}' \cup \{M\}$, wobei $\mathcal{P}' \in$ SGMod($\mathfrak{A}$) und M ein beliebiger vollständig spezifizierter Modul über $\mathfrak{A}$ mit
$(\forall M' \in \mathcal{P}')\mathrm{nam}(M') \neq \mathrm{nam}(M) \wedge$
$(\forall(\mathrm{name},\mathrm{est},\mathrm{ast}) \in \mathrm{GPA}(M))((\exists M' \in \mathcal{P}')(\mathrm{nam}(M') = \mathrm{name} \wedge$
$\mathrm{est}(M') = \mathrm{est} \wedge \mathrm{ast}(M') = \mathrm{ast})$,
gilt:
$\mathcal{P} \in$ SGMod($\mathfrak{A}$).

Def. 30

Sei $\mathcal{P} \in$ SGMod($\mathfrak{A}$).

a) $\leq'_{\mathcal{P}} \leftrightharpoons \{(M_1,M_2) \mid M_1,M_2 \in \mathcal{P} \wedge (M_1 \neq M_2 \rightarrow \exists\, x,y(\mathrm{nam}(M_1),x,y) \in \mathrm{GPA}(M_2))\}$.

b) $\leq_{\mathcal{P}} \leftrightharpoons \bigcup_{i \in N_0} (\leq'_{\mathcal{P}})^i$ (transitive Hülle von $\leq'_{\mathcal{P}}$)

mit $(\leq'_{\mathcal{P}})^0 \leftrightharpoons \{(M,M) \mid M \in \mathcal{P}\}$.

Kor. 31

Sei $\mathcal{P} \in$ SGMod($\mathfrak{A}$).
$\leq_{\mathcal{P}}$ ist Halbordnung auf $\mathcal{P}$.

Def. 32

a) $\mathcal{P} \in SGMod(\mathfrak{A})$ heißt Programm über $\mathfrak{A}$, wenn $(\mathcal{P}, \leq_{\mathcal{P}})$ ein größtes Element besitzt.

b) Sei $\mathcal{P} = \{M_1, M_2, \ldots, M_n\}$ ein Programm über $\mathfrak{A}$ und M_1 das größte Element von $(\mathcal{P}, \leq_{\mathcal{P}})$.

M_1 heißt Hauptprogramm von $\mathcal{P}$ und

$M_2, M_3, \ldots, M_n$ heißen globale Unterprogramme von $\mathcal{P}$.

c) Sei $\mathcal{P}$ ein Programm über $\mathfrak{A}$ und M ein partiell spezifizierter Modul oder das Hauptprogramm eines $P \in \mathcal{P}$.
$P' \in \mathcal{P}$ mit $\exists\, x,y(nam(P'),x,y)) \in GPA(M)$ heißt globales Unterprogramm von M.
Ein lokales Unterprogramm U von P heißt lokales Unterprogramm von M, wenn $nam(U) \in LPA(M)$.

Bemerkung

Die Bezeichnungen "Hauptprogramm" und "Unterprogramm" werden in Abhängigkeit vom Kontext in unterschiedlicher Bedeutung verwendet: Zunächst einmal wird bei vollständigen lokalen Unterprogramm-Systemen zwischen Hauptprogramm und Unterprogrammen unterschieden, dann zwischen dem Hauptprogramm und den lokalen Unterprogrammen eines vollständig spezifizierten Moduls, anschließend zwischen dem Hauptprogramm und den globalen Unterprogrammen eines Programms über $\mathfrak{A}$ und schließlich noch zwischen den globalen und lokalen Unterprogrammen bei partiell spezifizierten Moduln und Hauptprogrammen.

3.1.1.3 Graphentheoretische Modellbildung

$\mathfrak{P} = \{M_1, M_2, \ldots, M_{n(\mathfrak{P})}\}$ sei im folgenden ein beliebiges Programm-System über $\mathfrak{A}$, $M_g \in \mathfrak{P}$ beliebig und $M = M_g$ oder M ist das Hauptprogramm oder ein lokales Unterprogramm von M_g oder M ist ein beliebiger partiell-spezifizierter Modul von M_g oder ein paralleler Zweig eines parallelen Moduls von M_g oder der Schleifenkörper eines Schleifen-Moduls von M_g.

Def. 1

Eine Zeichenkette w aus M heißt:

i) Modulkopf, wenn

w ist Modulkopf von M_g oder Modulkopf eines kommentierten, funktions-spezifizierten oder lokal-aufrufbaren Moduls.

ii) Verallgemeinerte Wertzuweisung, wenn
w Assignation, lokaler oder globaler Prozeduraufruf oder w = "p-end (zahl) res wz" mit zahl $\in$ NNam(A) und wz $\in$ W(A').

iii) Zweigmarkierung, wenn
w = "B :" mit B Gleichung über A oder
w = "c-n :" mit n $\in$ NNam(A) oder
w = "otherwise :" oder
w = "p-n :" mit n $\in$ NNam(A).

iv) Parallelverzweigung, wenn w = "para(zahl){Φ}"
mit zahl $\in$ NNam(A) und $\Phi \in$ Form(A').

v) Dezentrale Verzweigung, wenn w = "if B"
mit B ist Gleichung über A.

vi) Zentrale Verzweigung, wenn
w = "case mod($B_1, B_2, \ldots, B_n$)" mit
mod $\in \{\Lambda, \text{one}\}$ und $(\forall\ 1 \leq i \leq n)$ B_i ist Gleichung über A.

vii) Einfachzweig-Abschluß, wenn w = "exit".

viii) Parallelzweig-Abschluß, wenn w = "p-exit $\{\Phi\}$"
mit $\Phi \in$ Form(A').

ix) Schleifenrücksprung, wenn w = "repeat".

x) Schleifeneingang, wenn w = "loop$\{\Phi\}$"
mit $\Phi \in$ Form(A').

xi) Einfachzweig-Sammelpunkt, wenn w = "end".

xii) Parallelzweig-Sammelpunkt, wenn w = "p-end(zahl)res wz"
mit zahl $\in$ NNam(A) $\wedge$ wz $\in$ W(A').

xiii) Prozedurende, wenn w = "$\{\Phi\}$fini".

xiv) Einfache Zusicherung, wenn w = "$\{\Phi\}$" und
w nicht Teil eines Prozedurendes, eines Schleifeneingangs, einer Parallelverzweigung oder eines Parallelzweig-Abschlusses ist.

xv) Verallgemeinerte Zusicherung, wenn w eine einfache Zusicherung oder ein Prozedurende, ein Schleifeneingang, eine Parallelverzweigung, ein Parallelzweig-Abschluß oder eine Zweigmarkierung der Form "p-n".ist.

Bemerkung

Jeder Zweigmarkierung der Form "p-n" wird eine Formel Φ als Zusicherung zugeordnet und zwar diejenige, die in der direkt vorangehenden Parallelverzweigung angegeben ist.

Def. 2

Eine Zeichenkette w aus M heißt elementare Moduloperation von M, wenn gilt:

i) w ist Modulkopf oder
ii) w ist verallgemeinerte Wertzuweisung oder
iii) w ist Zweigmarkierung oder
iv) w ist Parallelverzweigung oder
v) w ist dezentrale Verzweigung oder
vi) w ist zentrale Verzweigung oder
vii) w ist Einfachzweig-Abschluß oder
viii) w ist Parallelzweig-Abschluß oder
xi) w ist Schleifenrücksprung oder
x) w ist Schleifeneingang oder
xi) w ist Einfachzweig-Sammelpunkt oder
xii) w ist Parallelzweig-Sammelpunkt oder
xiii) w ist Prozedurende oder
xiv) w ist einfache Zusicherung.

Bemerkung

Während sich die Definition unserer Moduln an der Struktur möglicher bzw. zweckmäßiger Berechnungen sowie an der "Psychologie des Programmierens" orientiert, sind sowohl für den Prozeß des Programmierens als auch für den der Programm-Interpretation noch kleinere Bausteine unserer Moduln von Bedeutung, die wir im Hinblick auf unsere "operationelle Sicht" elementare Moduloperationen nennen. Auf diesen elementaren Moduloperationen wird die Semantik-Definition unserer Moduln aufbauen. Auch die angegebenen Methodiken zur Programm-Verifikation werden sich auf diese elementaren Moduloperationen beziehen.

Kor. 3

M besitzt eine eindeutig bestimmte Zerlegung in elementare Moduloperationen.

Bew.:

Induktion über den Aufbau von M.

Def. 4

a) $E_1E_2\ldots E_{n(M)}$ sei die Zerlegung von M in elementare Moduloperationen. Diese Zerlegung heißt operationelle Standardzerlegung von M.

b) End $\leftrightharpoons \{i \mid 1 \leq i \leq n(M) \wedge (E_i$ ist Prozedurende $\vee$ Parallelzweig-Abschluß $\vee\ i = n(M)\}$ ist die Menge der Endpunkte von M.

Bemerkung

Die folgenden Ausführungen beziehen sich auf die operationelle Standardzerlegung von M.

Def. 5

Ein knotenbewerteter, gerichteter Graph ohne Mehrfachkanten ist ein geordnetes Tripel $G = (P,R,\alpha)$, wobei P eine beliebige Menge, R eine zweistellige Relation in P und α eine Abbildung von P in eine Menge P_w.

Die Elemente von P heißen Punkte, die von R Kanten und die von P_w Punktwerte.

G heißt endlich, wenn P endlich ist.

Def. 6

Ein endlicher, knotenbewerteter, gerichteter Graph $G = (P,R,\alpha)$ ohne Mehrfachkanten heißt Standard-Strukturgraph von M ↯

Es gilt:

i) $P = \{1,2,\ldots,n(M)\} \wedge$

ii) $(\forall\, p \in P)\ \alpha(p) = E_p \wedge$

iii) $(\forall\, p \in P \setminus End)(\forall\, q \in P)((p,q) \in R$ gdw.

$((E_p$ ist Modulkopf $\vee$

E_p ist verallgemeinerte Wertzuweisung $\vee$

E_p ist Zweigmarkierung $\vee$

E_p ist Schleifeneingang $\vee$

E_p ist Einfachzweig-Sammelpunkt $\vee$

E_p ist einfache Zusicherung) $\wedge\ q = p+1) \vee$

$(E_p$ ist Parallelverzweigung $\wedge$

$q = \iota i(p \le i \le n(M) \wedge E_i$ ist Parallelzweig-Sammelpunkt $\wedge$

$(\forall\, p<j<i)\ E_j$ ist kein Parallelzweig-Sammelpunkt)

$(E_p$ ist dezentrale Verzweigung $\wedge\ (q = p+1 \vee$

$q = \iota i(p+1<i<n(M) \wedge E_{i-1}$ ist Einfachzweigabschluß $\wedge$

$(\forall p<j<i-1) E_j$ ist kein Einfachzweigabschluß))) $\vee$

$(E_p$ ist zentrale Verzweigung $\wedge\ (q = p+1 \vee$

$q \in \{i \mid p+3 \le i < n(M) \wedge E_{i-1}$ ist Einfachzweigabschluß $\wedge$

$(\forall\, p \le j \le i)\ E_j$ ist kein Einfachzweig-Sammelpunkt})) $\vee$

$(E_p$ ist Einfachzweig-Abschluß $\wedge$

$q = \iota i(p<i<n(M) \wedge E_i$ ist Einfachzweig-Sammelpunkt $\wedge$

$(\forall\, p<j<i)\ E_j$ ist kein Einfachzweig-Sammelpunkt)) $\vee$

$(E_p$ ist Schleifenrücksprung $\wedge$

$q = \iota i(1<i<p \wedge E_i$ ist Schleifeneingang $\wedge$

$(\forall\, i<j<p)\ E_j$ ist kein Schleifeneingang)).

Kor. 7

Es gibt genau einen Standard-Strukturgraph von M.

Bemerkung

$G = (P,R,\alpha)$ sei im folgenden der Standard-Strukturgraph von M. Besitzt M lokale Unterprogramme oder parallele Moduln, so zerfällt der Standard-Strukturgraph von M in mehrere Zusammenhangskomponenten. Beispiele für Standard-Strukturgraphen wurden bei der Syntax-Definition bereits skizziert.

Kor. 8

a) $(\forall\, q \in P)(\neg(\alpha(q)$ ist Einfachzweig-Sammelpunkt oder Schleifeneingang$) \rightarrow |\{p \mid (p,q) \in R\}| \leq 1)$.

b) $(\forall\, p \in P)(\alpha(p)$ ist Modulkopf oder verallgemeinerte Wertzuweisung oder Zweigmarkierung oder Parallelverzweigung oder Einfachzweigabschluß oder Schleifenrücksprung oder Schleifeneingang oder Einfachzweig-Sammelpunkt oder Parallelzweig-Sammelpunkt oder einfache Zusicherung
$\rightarrow |\{q \mid (p,q) \in R\}| = 1)$

Bemerkung

Die einzigen Programmstellen, an denen verschiedene Berechnungen "zusammentreffen" können sind Einfachzweig-Sammelpunkte und Schleifeneingänge. Diese Einschränkung ist - wie nicht nur die goto-Diskussion, sondern auch die nachfolgende Theoriebildung zeigen - für eine Methodik der Programmierung hilfreich und zweckmäßig.
An Parallelzweig-Sammelpunkten treffen nicht "verschiedene" Berechnungen aufeinander, sondern "Teilberechnungen" einer Berechnung.

Nach b) ist für einen Großteil der Punkte aus P die Relation R rechtseindeutig, d. h. eine Funktion. Zur Kennzeichnung des Bildes eines Punktes p unter R können wir deshalb auch R(p) schreiben.

Def. 9

Sei $K = (p_o, p_1, \ldots, p_m)$ mit $m \geq 1$ eine endliche nichtleere Folge mit $p_i \in P$ für $0 \leq i \leq m$.

a) K heißt einfacher, gerichteter Kantenzug in G $\leftrightarrow$

$(\forall\ 0 \leq i \leq m-1)(p_i, p_{i+1}) \in R \wedge$

$(\forall\ 0 < i < j < m)\ p_i \neq p_j$.

Für b) und c) sei K ein einfacher gerichteter Kantenzug in G.

b) $\mathrm{anf}(K) \leftrightarrow p_o$ heißt Anfangspunkt, $\mathrm{end}(K) \leftrightarrow p_m$ Endpunkt, $l(K) \leftrightarrow m$ Länge und $P(K) \leftrightarrow \{p_o, p_1, \ldots, p_m\}$ Punktmenge von K.

c) Sei $p \in P$ beliebig.

$$\mathrm{anf}(K,p) \leftrightarrow \begin{cases} \emptyset, \text{ falls } p \notin P(K) \vee (p = p_o \wedge p_m \neq p_o) \\ (p_o, p_1, \ldots, p_{m'}) \text{ mit } p_{m'} = p \wedge \begin{cases} 0 \leq m' \leq m \text{ falls } p_o \neq p_m \\ 0 < m' \leq m \text{ falls } p_o = p_m \end{cases} \text{ sonst.} \end{cases}$$

heißt Anfangsstück von K bis p.

$$\mathrm{end}(K,p) \leftrightarrow \begin{cases} \emptyset, \text{ falls } p \notin P(K) \vee (p = p_m \wedge p_o \neq p_m) \\ (p_{m'}, p_{m'+1}, \ldots, p_m) \text{ mit } p_{m'} = p \wedge \begin{cases} m' \leq m, \text{falls } p_o \neq p_m \\ m' < m, \text{falls } p_o = p_m \end{cases} \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{sonst} \end{cases}$$

heißt Endstück von K ab p.

Lemma 10

Für beliebige einfache, gerichtete Kantenzüge K und K' in G mit $\mathrm{anf}(K) = \mathrm{anf}(K')$ und $\mathrm{end}(K) = \mathrm{end}(K')$ gilt:
Wenn für alle $p \in P(K)$ $\alpha(p)$ kein Einfachzweig-Sammelpunkt ist, so gilt $K = K'$.

Bew.: Mit Kor. 8.

Def. 11

a) $ZS \leftrightharpoons \{p \in P \mid \alpha(p)$ ist verallgemeinerte Zusicherung$\}$.

b) Sei $p \in ZS$ beliebig.

$\psi(p) \leftrightharpoons \iota\Phi\ (\Phi \in Form(A') \wedge (\Phi$ kommt in $\alpha(p)$ vor $\vee$
$(\alpha(p)$ ist Zweigmarkierung $\wedge$
$(\exists\ 1 \leq q < p)(\alpha(q)$ ist Parallelverzweigung der Form <u>para</u>(zahl)$\{\Phi\} \wedge$
$(\forall\ q < q' < p)\ \alpha(q')$ ist keine Parallelverzweigung))))).

Bemerkung

ZS bezeichnet die Menge der Punkte, denen eine Zusicherung zugeordnet ist und die Funktion $\psi(p)$ stellt uns jeweils die Zusicherung am Punkt p zur Verfügung.

Def. 12

Seien $p,q \in P$ beliebig.

$K(p,q) = \{K \mid K$ ist einfacher gerichteter Kantenzug in $G \wedge$
$anf(K) = p \wedge end(K) = q \wedge (\forall\ p' \in P(K) \setminus \{p,q\}) p' \notin ZS\}$.

Def. 13

Sei $p \in P$ beliebig.

$ZV(p) \leftrightharpoons \{j \mid j \in ZS \wedge K(j,p) \neq \emptyset\}$

$ZN(p) \leftrightharpoons \{j \mid j \in ZS \wedge K(p,j) \neq \emptyset\}$

Bemerkung

ZV(p) dient zur Beschreibung derjenigen <u>Z</u>usicherungen, die einem Punkt p <u>v</u>orangehen und ZN(p) zur Beschreibung derjenigen <u>Z</u>usicherungen, die einem Punkt p <u>n</u>achfolgen.
$K \in K(p,q)$ ist ein Kantenzug von p nach q, der über keinen Zusicherungspunkt (Punkt aus ZS) führt.

3.1.1.4 Semantik

$\mathcal{P} = \{M_1, M_2, \ldots, M_{n(\mathcal{P})}\}$ sei weiterhin ein beliebiges Programm-System über $\mathfrak{A}$, $M_g \in \mathcal{P}$ beliebig und $M = M_g$ oder M ist das Hauptprogramm oder ein lokales Unterprogramm von M_g oder M ist ein beliebiger partiell-spezifizierter Modul von M_g oder ein paralleler Zweig eines parallelen Moduls von M_g oder der Schleifenkörper eines Schleifen-Moduls von M_g.
$E_1 E_2 \ldots E_{n(M)}$ sei die operationelle Standardzerlegung von M und $G = (P, R, \alpha)$ der Standard-Strukturgraph von M.
J sei die vollständige Interpretation des Berechnungsalphabets und J' die vollständige Interpretation des Verifikationsalphabets von M_g.

Def. 1

Ein Berechnungszustand von M ist ein Paar (i, σ) mit $i \in P \wedge \sigma \in \Sigma_J$.

Def. 2

Sei $z = (i, \sigma)$ ein beliebiger Berechnungszustand von M.

z ist abblockend (kurz $z \vdash\!\!\!- \Omega$) $\Leftrightarrow$
Eine der folgenden Bedingungen ist erfüllt:

i) $\alpha(i)$ ist Assignation der Form

$$\underline{\text{assi}}((t_1 \Rightarrow (v_{11}, \ldots, v_{1n_1})), \ldots, (t_n \Rightarrow (v_{m1}, \ldots, v_{mn_m})))$$

und $(\exists\, 1 \leq j \leq m)\ (t_j \in \text{Term}(A) \wedge t_j^{J,\sigma} = \Omega)$.

ii) $\alpha(i)$ ist lokaler Prozeduraufruf der Form

<u>lcall</u> name <u>res</u> $((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$

und es gilt:

$$\Phi_A(M')^{J',\sigma} \neq W \vee$$

$$(\exists\, \sigma' \in \Sigma_{J'})(\Phi_E(M')^{J',\sigma'} = W \wedge (\exists\, 1 \leq j \leq m)(t_j \in (\mathrm{Term}(A') \wedge$$

$$(t_j^{J',\sigma'} = \Omega \vee t_j^{J',\sigma'} \neq (v_{j1},\ldots,v_{jn_j})^{J',\sigma'}))),$$

wobei M' dasjenige lokale Unterprogramm von M_g mit nam(M') = name.

iii) $\alpha(i)$ ist globaler Prozeduraufruf der Form

<u>gcall</u> name$((t'_1,\ldots,t'_{m'}),(v'_1,\ldots,v'_{n'}))$

<u>res</u>$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$,

und es gilt:

$$(\exists\, 1 \leq j \leq m')\ t_j'^{J,\sigma} \notin J'_{M'}(b(e_j,J'_{M'}))\setminus\{\Omega\} \vee$$

$$\Phi_A(M')^{J'_{M'},\sigma'} \neq W \vee$$

$$(\exists\, \sigma'' \in \Sigma_{J'_{M'}})(\Phi_E(M')^{J'_{M'},\sigma''} = W \wedge (\forall\, 1 \leq j \leq m')\sigma''(e_j) = t_j'^{J,\sigma} \wedge$$

$$((\exists\, 1 \leq j \leq n')\ a_j^{J'_{M'},\sigma''} \notin J(b(v'_j,J))\setminus\{\Omega\} \vee (\exists\, 1 \leq j \leq m)$$

$$(t_j \in \mathrm{Term}(A') \wedge (t_j^{J',\sigma'''} = \Omega \vee t_j^{J',\sigma''} \neq (v_{j1},\ldots,v_{jn_j})^{J,\sigma'''}))))),$$

wobei M' dasjenige globale Unterprogramm von M_g mit nam(M') = name,

O.B.d.A. wurde vorausgesetzt:

input(M') = $(e_1,e_2,\ldots,e_{m'})$[1],

output(M') = $(a_1,a_2,\ldots,a_{n'})$,

$J'_{M'}$ ist die durch M' gegebene vollständige Interpretation des Verifikationsalphabets von M',

---

1) Per Definition von $\mathcal{P}$ gilt est(M') = m' und ast(M') = n'.

$$\sigma' = \left\langle x \mapsto \begin{cases} t_1'^{J,\sigma} & \text{falls } x = e_1 \\ \vdots & \\ t_m'^{J,\sigma} & \text{falls } x = e_{m'} \\ \Omega & \text{sonst} \end{cases} \mid x \in Var(M') \right\rangle^{2)}$$

und

$$\sigma''' = \left\langle x \mapsto \begin{cases} \sigma(x) & \text{falls } x \notin \{v_1', \ldots, v_{n'}'\} \\ \sigma''(a_1) & \text{falls } x = v_1' \\ \vdots & \\ \sigma''(a_{n'}) & \text{falls } x = v_{n'}' \end{cases} \mid x \in Var(M) \right\rangle$$

iv) $\alpha(i)$ ist Parallelzweig-Sammelpunkt der Form

$$\underline{\text{p-end}}(\text{zahl})\underline{\text{res}}((t_1 \Rightarrow (v_{11}, \ldots, v_{1n_1})), \ldots, (t_m \Rightarrow (v_{m1}, \ldots, v_{mn_m})))$$

und es gilt:

$$(\exists\, \sigma' \in \Sigma_{J'})(\Phi(M',1)^{J',\sigma_1'} = W \wedge \Phi(M',2)^{J',\sigma_2'} = W \wedge \ldots \wedge \Phi(M',fN(A)(zahl))^{J',\sigma_{fN(A)(zahl)}'} = w \wedge (\exists\, 1 \leq j \leq m)(t_j \in Term(A') \wedge (t_j^{J',\sigma'} = \Omega \vee t_j^{J',\sigma'} \neq (v_{j1}, \ldots, v_{jn_j})^{J',\sigma'})))$$

wobei M' derjenige parallele Modul, in dem $\alpha(i)$ vorkommt ($\Phi(M',j)$ ist damit für $1 \leq j \leq fN(A)(zahl)$ die Endzusicherung des j-ten parallelen Zweiges von M') und

$$\sigma_j' = \left\langle x \mapsto \begin{cases} \sigma'(x) & \text{falls } x \in MVar(M',j) \\ \sigma(x) & \text{sonst} \end{cases} \mid x \in Var(M_g) \right\rangle$$

---

1) σ' und σ''' beschreiben die Parameterübergabe beim Prozeduraufruf ("call by value").

v) $\alpha(i)$ ist dezentrale Verzweigung der Form

<u>if</u> B

und es gilt: $B^{J,\sigma} = \Omega$.

vi) $\alpha(i)$ ist zentrale Verzweigung der Form

<u>case</u> <u>one</u> $(B_1, B_2, \ldots, B_n)$

und es gilt:

$(\exists\ 1 \leq j, j' \leq n)(j \neq j' \wedge B_j^{J,\sigma} = B_{j'}^{J,\sigma} = W)$.

<u>Bezeichnung</u>

Ist z ein Berechnungszustand von M, so schreiben wir anstelle $\neg(z \vdash \Omega)$ kurz $z \nvdash \Omega$.

<u>Def. 3</u>

Sei $z = (i,\sigma)$ ein beliebiger Berechnungszustand von M mit $z \nvdash \Omega$ und $i \notin$ End.
Die <u>Modul-Berechnungsfunktion modfu</u> = (indexfu, belegfu) ordnet z einen neuen Berechnungszustand
$z' = \text{modfu}(z) = (\text{indexfu}(z), \text{belegfu}(z))$ zu,
der sich folgendermaßen ergibt:

i) Falls $\alpha(i)$ ein Modulkopf, eine Zweigmarkierung, eine Parallelverzweigung, ein Einfachzweig-Abschluß, ein Schleifenrücksprung, ein Schleifeneingang, ein Einfachzweig-Sammelpunkt, ein Parallelzweig-Sammelpunkt oder eine einfache Zusicherung, so gilt:

$$z' \Leftarrow (R(i), \sigma).$$

ii) Falls $\alpha(i)$ verallgemeinerte Wertzuweisung und

$$((t_1 \Rightarrow (v_{11}, \ldots, v_{1n_1})), (t_2 \Rightarrow (v_{21}, \ldots, v_{2n_2})), \ldots,$$
$$(t_m \Rightarrow (v_{m1}, \ldots, v_{mn_m})))$$

die in $\alpha(i)$ vorkommende Wertzuweisung, so gilt:

$z' \leftrightharpoons (i+1,\sigma')$ mit

$$\sigma' \leftrightharpoons \left\langle x \mapsto \begin{cases} \left.\begin{array}{l} P_1^{n_1}(t_1^{J',\sigma}),\ \text{falls } x = v_{11} \\ \vdots \qquad\qquad \vdots \\ P_{n_1}^{n_1}(t_1^{J',\sigma}),\ \text{falls } x = v_{1n_1} \end{array}\right\} \wedge\ t_1 \neq \underline{\text{frei}} \\ \Omega,\ \text{falls } x \in \{v_{11},\ldots,v_{1n_1}\} \wedge\ t_1 = \underline{\text{frei}} \\ \vdots \qquad\qquad \vdots \\ \left.\begin{array}{l} P_1^{n_m}(t_m^{J',\sigma}),\ \text{falls } x = v_{m1} \\ \vdots \qquad\qquad \vdots \\ P_{n_m}^{n_m}(t_m^{J',\sigma}),\ \text{falls } x = v_{mn_m} \end{array}\right\} \wedge\ t_m \neq \underline{\text{frei}} \\ \Omega,\ \text{falls } x \in \{v_{m1},\ldots,v_{mn_m}\} \wedge\ t_m = \underline{\text{frei}} \\ \sigma(x) \qquad \text{sonst.} \end{cases} \mid x \in \mathrm{Var}(M) \right\rangle$$ [1]

iii) Falls $\alpha(i)$ eine dezentrale Verzweigung der Form

<u>if</u> B, so gilt: $z' \leftrightharpoons (i',\sigma)$ mit

$$i' \leftrightharpoons \begin{cases} i+1, & \text{falls } B^{J,\sigma} = W \\ \iota j((i,j) \in R \wedge j \neq i+1) & \text{sonst.} \end{cases}$$

iv) Falls $\alpha(i)$ eine zentrale Verzweigung der Form

<u>case</u> $\mathrm{mod}(B_1,B_2,\ldots,B_n)$,

so gilt: $z' \leftrightharpoons (i',\sigma)$ mit

$$i' \leftrightharpoons \iota j((i,j) \in R \wedge |\{j' \mid i<j'<j \wedge (i,j') \in R\}| =$$
$$\iota k((k = n \wedge (\forall\ 1 \leq k' \leq k)\ B_{k'}^{J,\sigma} \neq W) \vee$$
$$(0 \leq k < n \wedge B_{k+1}^{J,\sigma} = W \wedge (\forall\ 1 \leq k' \leq k) B_{k'}^{J,\sigma} \neq W))).$$

<u>Bemerkung</u>

Falls $i \in \mathrm{End}$, d. h. i ein Endpunkt von M_g ist, so ist $\mathrm{modfu}(i,\sigma)$ undefiniert.

---

1) $P_i^n(x_1,x_2,\ldots,x_n) = x_i$ (i-te Projektion).

Def. 4

a) Ist z ein beliebiger Berechnungszustand von M und ist modfu(z) definiert, so heißt modfu(z) Folgezustand von z.

b) Eine endliche Folge $(z_j | j = 1,2,\ldots,m)$ von Berechnungszuständen von M mit $m \geq 2 \wedge z_{j+1} = \text{modfu}(z_j)$ für $j = 1,2,\ldots,m-1$ heißt Berechnung auf M.

Def. 5

Sei $B = (z_j | j = 1,2,\ldots,m)$ eine Berechnung auf M.

anf(B) $\leftrightharpoons z_1$ heißt Anfangszustand,

end(B) $\leftrightharpoons z_m$ Endzustand und

l(B) $\leftrightharpoons$ m Länge von B.

Def. 6

Sei (i,σ) ein beliebiger Berechnungszustand von M

M terminiert bezüglich (i,σ) $\leftrightharpoons$

$\exists B(B \text{ ist Berechnung auf } M \wedge \text{anf}(B) = (i,\sigma) \wedge P_1^2(\text{end}(B)) \in \text{End})$.

Bemerkung

Da modfu eine Funktion ist, gibt es zu jedem Berechnungszustand z, bezüglich dem M terminiert, genau eine Berechnung auf M mit anf(B) = z und $P_1^2(\text{end}(B)) \in \text{End}$.

Def. 7

Sei (i,σ) ein beliebiger Berechnungszustand von M und M terminiere bezüglich (i,σ).

$$\mathrm{val}(M,(i,\sigma)) \leftrightharpoons \iota\sigma'(\sigma' \in \Sigma_J \wedge \exists B(B \text{ ist Berechnung auf } M \wedge$$
$$\mathrm{anf}(B) = (i,\sigma) \wedge P_1^2(\mathrm{end}(B)) \in \mathrm{End} \wedge P_2^2(\mathrm{end}(B)) = \sigma'))$$

heißt Wert von M bezüglich (i,σ).

Def. 8

$$f(M) \leftrightharpoons \left\langle \sigma \mapsto \begin{cases} \mathrm{val}(M,(1,\sigma)), & \text{falls } \sigma \neq \Omega \wedge M \text{ bezüglich } (1,\sigma) \text{ terminiert} \\ \Omega & \text{sonst} \end{cases} \mid \sigma \in \Sigma_J \cup \{\Omega\} \right\rangle$$

heißt potentielle Bedeutung von M.

Def. 9

Sei $\Phi \in \mathrm{Form}(A')$ beliebig.

$$f(M,\Phi) \leftrightharpoons \left\langle \sigma \mapsto \begin{cases} f(M)(\sigma), & \text{falls } \Phi^{J',\sigma} = W \\ \Omega & \text{sonst} \end{cases} \mid \sigma \in \Sigma_J \cup \{\Omega\} \right\rangle$$

heißt Bedeutung von M bezüglich Φ.

Bemerkung

Unsere operationelle Semantik ermöglicht also nicht nur die Präzisierung der intuitiven Semantik, die z. B. ein FORTRAN- oder ALGOL-Programmierer seinen Anweisungen zugrundelegt, sowie die Präzisierung des bei Testläufen verwendeten Berechnungsbegriffs, sondern gestattet auch die Bedeutung eines Moduls als mathematische Funktion zu definieren und damit an die auf einer denotationellen Semantik aufbauenden Theoriebildung anzuknüpfen.

Def. 10

Eine Berechnung $B = (z_j \mid j = 1,2,\ldots,m)$ auf M heißt direkt $\leftrightharpoons$ $(\forall\ 1<j<m) P_1^2(z_j) \in ZS$.

Lemma 11

Seien z,z' beliebige Berechnungszustände von M.
Es gibt höchstens eine direkte Berechnung B auf M mit anf(B) = z und end(B) = z'.

Beweis

Folgt aus der Eigenschaft, daß modfu Funktion ist, sowie dem Aufbau von M (Existenz von Schleifenzusicherungen).

Def. 12

Sei $B = (z_j \mid j = 1,2,\ldots,m)$ eine direkte Berechnung auf M.

a) $pf(B) \leftrightharpoons (P_1^2(z_j) \mid j = 1,2,\ldots,m)$

heißt der zu B gehörende Kantenzug.

b) Sei $p \in P(pf(B))$ beliebig.

$$z(B,p) \leftrightharpoons \iota z \left(\left\{ \begin{array}{l} z \in \{z_j \mid 1 \leq j \leq m\}, \text{ falls } P_1^2(z_1) \neq P_1^2(z_m) \\ z \in \{z_j \mid 2 \leq j \leq m\} \text{ sonst} \end{array} \right\} \wedge P_1^2(z) = p \right)$$

heißt Berechnungszustand von B an der Stelle p.

Kor. 13

Sei B eine direkte Berechnung auf M .

$pf(B) \in K(P_1^2(anf(B)), P_1^2(end(B)))$.

Bemerkung

Bezüglich der semantischen Analyse (Programm-Verifikation) besitzen direkte Berechnungen eine ausgezeichnete Bedeutung.

3.1.2 Verifikation von Programm-Systemen

Entsprechend dem modularen Aufbau unserer Programm-Systeme zerteilt sich die Verifikation eines Programm-Systems auf die Verifikation seiner Bausteine, d. h. zunächst auf die Verifikation vollständig spezifizierter Moduln.
Die Verifikation vollständig spezifizierter Moduln besteht dann wiederum weitgehend aus der Verifikation der Bausteine vollständig spezifizierter Moduln, d. h. zunächst aus der Verifikation des Hauptprogramms und der lokalen Unterprogramme eines vollständig spezifizierten Moduls. Eine letzte Gliederung der Verifikationsaufgabe ergibt sich noch bei im Hauptprogramm oder in einem lokalen Unterprogramm vorkommenden funktionsspezifizierten Moduln, parallelen Moduln oder Schleifen-Moduln, die den Verifikationsprozeß in eine "Hauptrechnung" und in "Nebenrechnungen" zerlegen.

Neben einer Modularisierung des Verifikationsprozesses streben wir auch eine Begründung und Verbesserung existierender Verifikationskalküle und -methoden auf der Basis unserer operationellen (Berechnungs-)Semantik an. Dabei wird zunächst zwischen Methoden zum Nachweis der partiellen Korrektheit und Methoden zum Nachweis der Programmterminierung unterschieden.

Manna [1974] nennt ein Programm bereits dann partiell korrekt, wenn jede zulässige Berechnung, die vom Programmanfang zum Programmende führt, mit einer Belegung endet, welche die dem Programmende zugeordnete Zusicherung erfüllt. Will man diese Eigenschaft eines Programms beweisen, so benötigt man in der Regel jedoch nicht nur Anfangs- und Endzusicherungen, sondern zumindest noch Schleifenzusicherungen. Außerdem sollten bei einem korrekten Programm nicht nur im Hinblick auf nachfolgende Änderungen zumindest alle Unterprogramme korrekt (spezifiziert) sein. Darüber hinaus ist es vorteilhaft, die Eigenschaft des Nichtabblockens von Berechnungen (Vermeidung von Division

durch Null, Überlauf u. ä.) zusammen mit der Transformation von Zusicherungen - und nicht etwa beim Nachweis der Endlichkeit der Schleifendurchläufe ("Terminierung") - zu zeigen. Wir werden also, aufbauend auf unserer operationellen Semantik, nicht nur von einem präzisierten, sondern auch von einem enger gefaßten Begriff der partiellen Korrektheit ausgehen.

Bei den Methoden zum Nachweis der partiellen Korrektheit von Programm-Moduln unterscheiden wir zwischen Verifikationsmethoden während und nach der Programmerstellung.

Bei der Verifikation während der Programmerstellung geht es im wesentlichen um die Erfüllung vorgegebener Formeln (pre-, post-conditions) durch geeignete Programmkonstruktionen. Je nachdem, ob die Programmkonstruktion von der Anfangszusicherung zur Endzusicherung oder umgekehrt von der Endzusicherung zur Anfangszusicherung vorangetrieben wird, benötigen wir die jeweils aktualisierte, entsprechend den bisher erstellten Anweisungen transformierte Anfangs- bzw. Endzusicherung, d. h. die "strongest post-condition" bzw. die "weakest pre-condition", wenn auf Abschwächungen (Folgerungen) verzichtet wird. Derartige Abschwächungen sind jedoch in der Regel bei der Programmkonstruktion notwendig bzw. hilfreich.
Die begrifflichen Bezugspunkte für die Programmverifikation während der Programmerstellung sind deshalb auch nicht die "strongest post-condition" und "weakest pre-condition" wie z. B. bei Dijkstra [1976] oder Emden [1976], sondern die Begriffe "vorwärtsgültige Formel am (Programm-)Punkt i" und "rückwärtsgültige Formel am (Programm-)Punkt i".
Das Ziel bei der Programmkonstruktion besteht damit darin, die Endzusicherung als vorwärtsgültige Formel bzw. die Anfangszusicherung als rückwärtsgültige Formel mithilfe geeignet gewählter Anweisungen "aufzubauen", d. h. mithilfe eines Kalküls abzuleiten.

Unsere Kalküle zur Transformation vorwärts- bzw. rückwärtsgültiger Formeln beziehen sich dabei im Unterschied zur z. B. Dijkstra'schen Prädikattransformation wp(S,Φ) nicht auf vollständige Anweisungen, sondern auf elementare Moduloperationen, ermöglichen also eine feinere Argumentation bei der Programm-Konstruktion.

Die Verifikation nach der Programmerstellung gliedert sich in den Nachweis, daß jeder einfache gerichtete Kantenzug partiell korrekt ist. In Anlehnung an King [1976] werden drei verschiedene Methoden für den Nachweis der partiellen Korrektheit von Kantenzügen betrachtet, die auch zur automatischen Erzeugung von Verifikations-Bedingungen im Rahmen von Programm-Verifikations-Systemen verwendet werden (z. B. King [1969], Henke/Luckham [1975]).

Ähnlich wie bei der partiellen Korrektheit läßt sich auch die globale Eigenschaft der Programm-Terminierung auf lokale Argumentationen zurückführen:
Ist ein Programm (bzw. Modul) M partiell korrekt und terminiert jeder Schleifen-Modul von M, so terminiert M.
Unsere Überlegungen zum Nachweis der Programm-Terminierung können sich damit auf die Frage nach der Terminierung von Schleifen-Moduln beschränken. Das hierbei entwickelte Verfahren hat gegenüber dem bisher üblichen Nachweis der Terminierung mittels wohlgeordneter (bzw. fundierter) Mengen (siehe z. B. Manna [1974] pp. 195) vorallem den Vorteil, daß es direkt auf dem Nachweis der partiellen Korrektheit aufbauen kann (die gegebene Schleifeninvariante ist der Anknüpfungspunkt) und eine Abschätzung der maximalen bzw. durchschnittlichen Programmlaufzeit ermöglicht.

Zwischen unserer Terminierungs-Methode und der Methode intermittierender Zusicherungen (Manna/Waldinger [1978]) besteht hierbei ein enger Zusammengang. Unsere Terminierungs-Methode zerlegt jede Schleifeninvariante in eine Disjunktion ($\Phi_{repeat} \vee \Phi_{end}$), wobei durch Φ_{repeat} unmittelbar zu einem Schleifenrücksprung (repeat) und durch Φ_{end} unmittelbar zu einem Schleifenende (end) führende Berechnungen charakterisiert sind. Bei der Programmausführung werden somit sowohl Φ_{repeat} als auch Φ_{end} nur manchmal (sometimes) erfüllt sein, d. h. Φ_{repeat} und Φ_{end} sind intermittierende Zusicherungen.
Ähnlich zur Problemstellung bei der Methode intermittierender Zusicherungen reduziert sich bei unserer Terminierungs-Methode der Nachweis der Terminierung eines Schleifen-Moduls auf den Nachweis, daß immer dann, wenn eine Berechnung Φ_{repeat} erfüllt, die Fortsetzung dieser Berechnung irgendwann ("at sometime"), wenn die Berechnung wieder zu dem Programmpunkt kommt, Φ_E erfüllt.

$\mathfrak{P} = \{M_1, M_2, \ldots, M_{n(\mathfrak{P})}\}$ sei weiterhin ein beliebiges Programm-System über $\mathfrak{A}$, $M_g \in \mathfrak{P}$ beliebig und $M = M_g$ oder M ist das Hauptprogramm oder ein lokales Unterprogramm von M_g oder M ist ein beliebiger funktions-spezifizierter Modul[1)] von M_g oder ein paralleler Zweig eines parallelen Moduls von M_g oder der Schleifenkörper eines Schleifen-Moduls von M_g.

$E_1 E_2 \ldots E_{n(M)}$ sei die operationelle Standardzerlegung von M und $G = (P, R, \alpha)$ der Standard-Strukturgraph von M.
J sei die vollständige Interpretation des Berechnungsalphabets und J' die vollständige Interpretation des Verifikationsalphabets von M_g.

---

1) Anstelle partiell-spezifizierter Moduln setzen wir funktions-spezifizierte Moduln voraus, da wir die Frage nach der Korrektheit nur für Programm-Moduln mit Anfangs- und Endzusicherungen stellen.

3.1.2.1 Partielle Korrektheit

3.1.2.1.1 Theoretische Grundlagen

Def. 1

M ist partiell korrekt $\leftrightarrows$

$$\forall B\ (B \text{ ist Berechnung auf } M \wedge P_1^2(\mathrm{anf}(B)) \in ZS \wedge$$

$$\psi(P_1^2(\mathrm{anf}(B)))^{J',P_2^2(\mathrm{anf}(B))} = W \rightarrow$$

$$(P_1^2(\mathrm{end}(B)) \notin \mathrm{End} \rightarrow \mathrm{end}(B) \not\mapsto \Omega) \wedge$$

$$(P_1^2(\mathrm{end}(B)) \in ZS \rightarrow \psi(P_1^2(\mathrm{end}(B)))^{J',P_2^2(\mathrm{end}(B))} = W)).$$

Def. 2

$\mathcal{P}$ ist partiell korrekt $\leftrightarrows$

$(\forall M_g \in \mathcal{P})$ M_g ist partiell korrekt.

Def. 3

Sei $\Phi \in \mathrm{Form}(A')$ und $i \in P$ beliebig.

a) Φ ist vorwärtsgültig bei i $\leftrightharpoons$

$$(\forall j \in ZV(i))\ \forall B(B \text{ ist direkte Berechnung auf } M \wedge$$
$$P_1^2(\mathrm{anf}(B)) = j \wedge \psi(j)^{J',P_2^2(\mathrm{anf}(B))} = W \rightarrow$$
$$(\forall k \in K(j,i))(\forall p \in P(k)\setminus\{i\})(\ \mathrm{anf}(k,p) \neq \emptyset \wedge \mathrm{anf}(k,p) = \mathrm{pf}(B,p)$$
$$\rightarrow z(B,p) \not\mapsto \Omega) \wedge$$
$$(P_1^2(\mathrm{end}(B)) = i \rightarrow \Phi^{J',P_2^2(\mathrm{end}(B))} = W)\ ).$$

b) Φ ist rückwärtsgültig bei i $\leftrightharpoons$

$$ZN(i) = \emptyset \vee$$
$$(\forall \sigma \in \Sigma_J)(\Phi^{J',\sigma} = W \rightarrow \exists B(B \text{ ist direkte Berechnung auf } M \wedge$$
$$\mathrm{anf}(B) = (i,\sigma) \wedge P_1^2(\mathrm{end}(B)) \in ZN(i) \wedge$$
$$\psi(P_1^2(\mathrm{end}(B)))^{J',P_2^2(\mathrm{end}(B))} = W)).$$

Bemerkung

Zusicherungen haben den Charakter von Hypothesen und vorwärts- bzw. rückwärts-gültige Formeln den wahrer Sätze. Wie im folgenden gezeigt wird, ist M genau dann partiell korrekt, wenn alle in M vorkommenden Zusicherungen vorwärts- bzw. rückwärts-gültig sind.

Lemma 4

M ist partiell korrekt gdw. gilt:

$$\forall B(B \text{ ist direkte Berechnung auf } M \wedge P_1^2(\mathrm{anf}(B)) \in ZS$$
$$\wedge\ \psi(P_1^2(\mathrm{anf}(B)))^{J',P_2^2(\mathrm{anf}(B))} = W \rightarrow$$
$$(P_1^2\ (\mathrm{end}(B)) \notin \mathrm{End} \rightarrow \mathrm{end}(B) \not\mapsto \Omega) \wedge$$
$$(P_1^2(\mathrm{end}(B)) \in ZS \rightarrow \psi(P_1^2(\mathrm{end}(B)))^{J',P_2^2(\mathrm{end}(B))} = W)).$$

Beweis

"$\Rightarrow$" Mit Definition 1.

"$\Leftarrow$"

Sei B eine beliebige Berechnung auf M mit $P_1^2(\mathrm{anf}(B)) \in ZS$

und $\psi(P_1^2(\mathrm{anf}(B)))^{J',P_2^2(\mathrm{anf}(B))} = W$.

Ist B eine direkte Berechnung, so bleibt nichts zu zeigen. Anderenfalls läßt sich B in eine eindeutig bestimmte Folge direkter Berechnungen $B_1, B_2, \ldots, B_n$ zerlegen mit $n \geq 2$ und $B = B_1\ B_2 \ldots B_n$ [1] wobei gilt:

$$(\forall\ 1 \leq j \leq n-1)\,(P_1^2(\mathrm{anf}(B_j)), P_1^2(\mathrm{end}(B_j)) \in ZS \wedge \psi(P_1^2(\mathrm{end}(B_j)))^{J',P_2^2(\mathrm{end}(B_j))} = W)$$

und

$$(P_1^2(\mathrm{end}(B_n)) \notin \mathrm{End} \rightarrow \mathrm{end}(B) \not\vdash \Omega) \wedge$$

$$(P_1^2(\mathrm{end}(B_n)) \in ZS \rightarrow \psi(P_1^2(\mathrm{end}(B_n)))^{J',P_2^2(\mathrm{end}(B_n))} = W).$$

Mit $\mathrm{end}(B_n) = \mathrm{end}(B)$ ergibt sich die partielle Korrektheit von M.

Lemma 5

Sei $i \in ZS$ beliebig.

a) Wenn gilt:
$(\forall j \in ZV(i))\ \psi(j)$ ist rückwärtsgültig bei j, so ist $\psi(i)$ vorwärtsgültig bei i.

b) Wenn gilt:
$(\forall j \in ZN(i))\ \psi(j)$ ist vorwärtsgültig bei j, so ist $\psi(i)$ rückwärtsgültig bei i.

---

1) D. h. mit $\mathrm{anf}(B_{j+1}) = \mathrm{end}(B_j)$ für $j = 1, \ldots, n-1$.

Beweis

a) Sei $j \in ZV(i)$ beliebig und B eine beliebige direkte Berechnung auf M mit $P_1^2(anf(B)) = j \wedge \psi(j)^{J',P_2^2(anf(B))} = W$.

Da $\psi(j)$ rückwärtsgültig bei j existiert eine direkte Berechnung B' auf M mit $anf(B') = anf(B) \wedge P_1^2(end(B')) \in ZN(j)$, d. h. B ist Anfangsstück von B' oder es gilt B = B'.

Somit gilt:

$(\forall k \in K(j,i))(\forall p \in P(k)\setminus\{i\})(anf(k,p) \neq \emptyset \wedge anf(k,p) = pf(B,p) \rightarrow z(B,p) \not\vdash \Omega)$.

Gilt $P_1^2(end(B)) = i$, so gilt B = B' und mit $\psi(P_1^2(end(B)))^{J',P_2^2(end(B))} = W$ gilt auch

$\psi(i)^{J',P_2^2(end(B'))} = W$.

Somit ist $\psi(i)$ vorwärtsgültig bei i.

b) Sei $ZN(i) \neq \emptyset$ und $\sigma \in \Sigma_J$ mit $\psi(i)^{J',\sigma} = W$ beliebig.

Zu zeigen:

$\exists B(B$ ist direkte Berechnung auf $M \wedge anf(B) = i,\sigma) \wedge$

$P_1^2(end(B)) \in ZN(i) \wedge \psi(P_1^2(end(B)))^{J',P_2^2(end(B))} = W)$.

Da für alle $j \in ZN(i)$ $\psi(j)$ vorwärtsgültig bei j, gilt für alle direkten Berechnungen B mit $anf(B) = (i,\sigma)$: B blockt nicht ab und führt schließlich zu einem Punkt $j \in ZN(i)$ mit $\psi(j)^{J',P_2^2(end(B))} = W$.

Satz 6

Die folgenden Aussagen sind äquivalent:

i) M ist partiell korrekt.

ii) $(\forall i \in ZS)(\psi(i)$ ist vorwärtsgültig bei $i)$.

iii) $(\forall i \in ZS)(\psi(i)$ ist rückwärtsgültig bei $i)$.

Beweis

i) $\Rightarrow$ ii) Mit Lemma 4.

ii) $\Rightarrow$ i):

Mit Lemma 4 ist für eine beliebige direkte Berechnung B auf M

mit $P_1^2(anf(B)) \in ZS \wedge \psi(P_1^2(anf(B)))^{J',P_2^2(anf(B))} = W$

zu zeigen:

$\alpha)$ $P_1^2(end(B)) \notin End \rightarrow end(B) \not\rightarrow \Omega$ und

$\beta)$ $P_1^2(end(B)) \in ZS \rightarrow \psi(P_1^2(end(B)))^{J',P_2^2(end(B))} = W.$

B verläuft auf einer Kante $k \in K(P_1^2(anf(B)),j)$ mit $j \in ZS$

Da $\psi(j)$ nach Voraussetzung vorwärtsgültig bei j gilt $\alpha)$ und $\beta)$.

ii) $\Leftrightarrow$ iii): Mit Lemma 5.

Bemerkung

Auf Satz 6 bauen die Methoden zum Nachweis der partiellen Korrektheit auf:
Die Eigenschaft der in Def. 1 präzisierten partiellen Korrektheit wird beim Verifikationsprozess in Abhängigkeit von der jeweiligen Verifikationsmethode (Vorwärts- bzw. Rückwärts-Entwicklung von Zusicherungen) dadurch gezeigt werden, daß die Vorwärts- bzw. Rückwärtsgültigkeit der Zusicherungen bewiesen wird.

3.1.2.1.2 Verifikationskalküle und -methoden

Def. 1

Seien $\Phi \in \mathrm{Form}(A')$ und $(i,j) \in R$ beliebig.

Die Vorwärtstransformation von Φ bezüglich (i,j) (kurz $\mathrm{vtraf}((i,j),\Phi)$) ist definiert durch:

i) Falls $\alpha(i)$ ein Modulkopf, eine Zweigmarkierung, eine Parallelverzweigung, ein Einfachzweig-Abschluß, ein Schleifenrücksprung, ein Schleifeneingang, ein Einfachzweig-Sammelpunkt oder eine einfache Zusicherung, so gilt:

$$\mathrm{vtraf}((i,j),\Phi) \leftrightharpoons \begin{cases} \Phi' & \text{falls in } \alpha(i) \text{ eine Zeichenkette } \{\Phi'\} \text{ mit } \Phi' \in \mathrm{Form}(A') \text{ vorkommt} \\ \Phi & \text{sonst.} \end{cases}$$

ii) Falls $\alpha(i)$ eine verallgemeinerte Wertzuweisung und

$$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$$

die in $\alpha(i)$ vorkommende Wertzuweisung, wobei $(\forall\ 1 \leq k \leq m)\ t_k \neq \underline{\text{frei}}$[1], so gilt:

$$\mathrm{vtraf}((i,j),\Phi) \leftrightharpoons$$

$$(\exists v'_{11},\ldots,v'_{mn_m})(\Phi(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}}) \wedge t_1(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}}) = (v_{11},\ldots,v_{1n_1}) \wedge$$

$$\ldots \wedge t_m(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}}) = (v_{m1},\ldots,v_{mn_m}))$$ [2]

---

1) Wir wollen uns auf die Darstellung dieses Falls beschränken. Der Fall $(\exists\ 1 \leq k \leq m)\ t_k = \underline{\text{frei}}$ ist weitgehend analog zu handhaben.

2) $\Phi(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}})$ bzw. $t(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}})$ ist diejenige Zeichenkette, die aus Φ bzw. t entsteht, wenn v_{ij} durch v'_{ij} ersetzt wird für $1 \leq i \leq m, 1 \leq j \leq n_i$).

wobei

$$(\forall\ 1 \le j \le m)(\forall\ 1 \le i \le n_j)(v'_{ji} \in Var(A') \wedge v'_{ji} \notin FVVar(M_g) \wedge$$

$$v'_{ji} \text{ bezgl. } \Phi \text{ neu}^{1)} \wedge b(v'_{ji}, J') = b(v_{ji}, J')).$$

iii) Falls $\alpha(i)$ eine dezentrale Verzweigung der Form <u>if</u> B, so gilt:

$$vtraf((i,j),\Phi) \leftrightharpoons \begin{cases} (\Phi \wedge B) & \text{falls } j = i+1 \\ (\Phi \wedge \neg B) & \text{sonst.} \end{cases}$$

iv) Falls $\alpha(i)$ eine zentrale Verzweigung der Form <u>case</u>$(B_1, B_2, \ldots, B_n)$, so gilt:

$$vtraf((i,j),\Phi) \leftrightharpoons \begin{cases} (\Phi \wedge B_1) & \text{falls } j = i+1 \\ (\Phi \wedge \neg B_1 \wedge B_2) & \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = 1 \\ \quad\vdots & \\ (\Phi \wedge \neg B_1 \wedge \neg B_2 \wedge \ldots \wedge \neg B_{n-1} \wedge B_n) & \\ \qquad \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = n-1 & \\ (\Phi \wedge \neg B_1 \wedge \ldots \wedge \neg B_n) & \text{falls } |\{j' \mid (i,j') \in R \\ & \qquad \wedge\ j' < j\}| = n. \end{cases}$$

v) Falls $\alpha(i)$ eine zentrale Verzweigung der Form <u>case</u> <u>one</u> $(B_1, B_2, \ldots, B_n)$, so gilt:

$$vtraf((i,j),\Phi) \leftrightharpoons \begin{cases} (\Phi \wedge B'_1) & \text{falls } j = i+1 \\ (\Phi \wedge B'_2) & \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = 1 \\ \quad\vdots & \\ (\Phi \wedge B'_n) & \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = n-1 \\ (\Phi \wedge \neg B_1 \wedge \neg B_2 \wedge \ldots \wedge \neg B_n) & \text{falls} \\ & |\{j' \mid (i,j') \in R \wedge j' < j\}| = n \end{cases}$$

wobei für alle $1 \le k \le n$: $B'_k \leftrightharpoons \neg B_1 \wedge \ldots \wedge \neg B_{k-1} \wedge B_k \wedge \neg B_{k+1} \wedge \ldots \wedge \neg B_n$.

---

1) D. h. v'_{ji} kommt in Φ weder frei noch gebunden vor.

Def. 2

Seien $\Phi \in \mathrm{Form}(A')$ und $i \in P$ beliebig.

i ist bezüglich Φ nicht abblockend (kurz $(i,\Phi) \not\vdash \Omega$) $\leftrightharpoons$

$(\forall\sigma \in \Sigma_J)(\Phi^{J',\sigma} = W \rightarrow (i,\sigma) \not\vdash \Omega)$.

Satz 3

(Kalkül zur Entwicklung vorwärtsgültiger Formeln)

Sei $i \in P$ beliebig und $\Phi \in \mathrm{Form}(A')$ eine beliebige bei i vorwärtsgültige Formel.

a) Für alle $\Phi' \in \mathrm{Form}(A')$ mit

$(\forall\sigma \in \Sigma_J)(\Phi^{J',\sigma} = W \rightarrow \Phi'^{J',\sigma} = W)$

gilt: Φ' ist vorwärtsgültig bei i.

b) Sei $j \in P$ beliebig mit $(i,j) \in R$.

b1) Ist $|\{i' \mid (i',j) \in R\}| = 1$ und ist i bezüglich Φ nicht abblockend, so gilt:
$\mathrm{vtraf}((i,j),\Phi)$ ist vorwärtsgültig bei j.

b2) Ist $|\{i' \mid (i',j) \in R\}| > 1$ und ist für alle k mit $(k,j) \in R$ Φ_k eine bei k vorwärtsgültige Formel und k bezüglich Φ_k nicht abblockend, so gilt:

$$\bigvee_{k \in \{i' \mid (i',j) \in R\}} \mathrm{vtraf}((k,j),\Phi_k)$$

ist vorwärtsgültig bei j.

Beweis:

a) Siehe Definition von "vorwärtsgültig".

b1) Sei $i' \in ZV(j)$ beliebig und B eine beliebige direkte Berechnung auf M mit $P_1^2(anf(B)) = i' \wedge$

$$\psi(i')^{J',P_2^2(anf(B))} = W.$$

Zu zeigen:

α) $(\forall k \in K(i',j))(\forall p \in P(k)\setminus\{j\})(anf(k,p) \neq \emptyset \wedge$
$anf(k,p) = pf(B,p) \rightarrow z(B,p) \not\mapsto \Omega)$ und

β) $P_1^2(end(B)) = j \rightarrow vtraf((i,j),\Phi)^{J',P_2^2(end(B))} = W.$

Fallunterscheidung:

i) α(i) ist verallgemeinerte Zusicherung, d. h. $i \in ZS$.

Da $|\{i''|(i'',j) \in R\}| = 1$ gilt $ZV(j) = \{i\}$.

Somit gilt $i' = i$, $K(i,j) = \{(i,j)\}$.

Da $anf((i,j),i) = \emptyset$ bleibt nur β) zu zeigen.

Da $P_2^2(anf(B)) = P_2^2(end(B))$ (Def. 3 von 3.1.1.4)

gilt $\psi(i)^{J',P_2^2(end(B))} = W.$

Mit $vtraf((i,j),\Phi) = \psi(i)$ gilt die Behauptung.

ii) α(i) ist keine verallgemeinerte Zusicherung, d. h. $i \notin ZS$.

Fallunterscheindung:

ii1) α(i) ist verallgemeinerte Wertzuweisung, wobei

$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$

die in α(i) vorkommende Wertzuweisung sei.[1)]

Mit $|\{i''|(i'',j) \in R\}| = 1$ gilt $ZV(j) = ZV(i)$.

---

1) Wir beschränken uns wieder auf den Fall, daß $(\forall\ 1 \leq k \leq m)\ t_k \neq$ <u>frei</u>.

α) Seien $k = (p_1, p_2, \ldots, p_n) \in K(i',j)$ und $p \in P(k) \setminus \{j\}$ beliebig mit $anf(k,p) \neq \emptyset$ und $anf(k,p) = pf(B,p)$.

Zu zeigen: $z(B,p) \not\vdash \Omega$.

Fallunterscheidung:

$p \neq p_{n-1}$: Da Φ nach Voraussetzung eine bei i vorwärtsgültige Formel gilt $z(B,p) \not\vdash \Omega$.

$p = p_{n-1}$: Nach Voraussetzung gilt $\Phi^{J',P_2^2(z(B,p))} = W$ und $p = i$.
Da i bezüglich Φ nicht abblockend gilt $z(B,p) \not\vdash \Omega$.

β) Sei $P_1^2(end(B)) = j$.

Zu zeigen: $vtraf((i,j),\Phi)^{J',P_2^2(end(B))} = W$.

Sei $\sigma \leftrightharpoons P_2^2(z(B,i))$.

Zu zeigen bleibt (Oberschelp [1974] S. 191):

$$(\exists d_{11},\ldots,d_{mn_m})((\Phi(\begin{smallmatrix} v_{11}\ldots v_{mn_m} \\ v'_{11}\ldots v'_{mn_m} \end{smallmatrix}) \wedge t_1(\begin{smallmatrix} v_{11}\ldots v'_{mn_m} \\ v'_{11}\ldots v'_{mn_m} \end{smallmatrix}) =$$

$$(v_{11},\ldots,v_{1n_1}) \wedge \ldots \wedge t_m(\begin{smallmatrix} v_{11}\ldots v_{mn_m} \\ v'_{11}\ldots v'_{mn_m} \end{smallmatrix}) =$$

$$(v_{m1},\ldots,v_{mn_m}))^{J',P_2^2(end(B))(\begin{smallmatrix} v'_{11}\ldots v'_{mn_m} \\ d_{11}\ldots d_{mn_m} \end{smallmatrix})} = W)$$

wobei $\sigma'(\begin{smallmatrix} v'_{11}\ldots v'_{mn_m} \\ d_{11}\ldots d_{mn_m} \end{smallmatrix})$ diejenige Belegung, die v'_{ij} mit d_{ij} belegt (für $1 \leq i \leq m, 1 \leq j \leq n_i$) und alle anderen Variablen wie σ'.

Für $d_{ij} \leftrightharpoons \sigma(v_{ij})$ $(1 \leq i \leq m, 1 \leq j \leq n_i)$ gilt:

$$(\Phi(\begin{smallmatrix} v_{11} \cdots v_{mn_m} \\ v'_{11} \cdots v'_{mn_m} \end{smallmatrix}))^{J',P_2^2(end(B))} (\begin{smallmatrix} v'_{11} \cdots v'_{mn_m} \\ d_{11} \cdots d_{mn_m} \end{smallmatrix}) = \Phi^{J',\sigma} = W$$

und

$$(t_i(\begin{smallmatrix} v_{11} \cdots v'_{mn_m} \\ v'_{11} \cdots v'_{mn_m} \end{smallmatrix}) = (v_{i1},\ldots,v_{in_i}))^{J',P_2^2(end(B))} (\begin{smallmatrix} v'_{11} \cdots v'_{mn_m} \\ d_{11} \cdots d_{mn_m} \end{smallmatrix})$$

$= W$ für $1 \leq i \leq m$, da $t_i^{J',\sigma} = (v_{i1},\ldots,v_{im_i})^{J',P_2^2(end(B))}$.

ii2) $\alpha(i)$ ist dezentrale Verzweigung der Form <u>if</u> B_1.

Es gilt $ZV(j) = ZV(i)$.

α) Analog zu ii1)α).

β) Fallunterscheidung:

β1): $j = i+1$.

Es gilt $B_1^{J,P_2^2(end(B))} = W$.

Mit $\Phi^{J',P_2^2(end(B))} = W$ gilt somit

$vtraf((i,j),\Phi)^{J',P_2^2(end(B))} = W$.

β2): $j \neq i+1$.

Da i bezüglich Φ nicht abblockend und

$B_1^{J,P_2^2(end(B))} \neq W$ gilt $B_1^{J,P_2^2(end(B))} = F$.

Mit $\Phi^{J',P_2^2(end(B))} = W$ gilt

$vtraf(i,j),\Phi)^{J',P_2^2(end(B))} = W$.

ii3) $\alpha(i)$ ist zentrale Verzweigung der Form
<u>case</u> $(B_1, B_2, \ldots, B_n)$.
Analog zu ii2).

ii4) $\alpha(i)$ ist zentrale Verzweigung der Form
<u>case</u> <u>one</u> $(B_1, B_2, \ldots, B_n)$.
Analog zu ii2).

ii5) $\alpha(i)$ ist keine verallgemeinerte Zusicherung, keine verallgemeinerte Wertzuweisung, keine dezentrale und keine zentrale Verzweigung.
Analog zu ii2) mit $\mathrm{vtraf}((i,j),\Phi) = \Phi$ und
$\Phi^{J',P_2^2(\mathrm{end}(B))} = W$.

b2) Sei $i' \in ZV(j)$ beliebig und B eine beliebige direkte Berechnung auf M mit $P_1^2(\mathrm{anf}(B)) = i'$ und
$\psi(i')^{J',P_2^2(\mathrm{anf}(B))} = W$.

Zu zeigen:

$\alpha)$ $(\forall k \in K(i',j))(\forall p \in P(k)\setminus\{j\})(\mathrm{anf}(k,p) \neq \emptyset \wedge$
$\mathrm{anf}(k,p) = \mathrm{pf}(B,p) \rightarrow z(B,p) \not\mapsto \Omega)$ und

$\beta)$ $P_1^2(\mathrm{end}(B)) = j \rightarrow \mathrm{vtraf}((i,j),\Phi)^{J',P_2^2(\mathrm{end}(B))} = W$.

zu $\alpha)$: Seien $k \in K(i',j)$ und $p \in P(k)\setminus\{j\}$ beliebig mit $\mathrm{anf}(k,p) \neq \emptyset$ und $\mathrm{anf}(k,p) = \mathrm{pf}(B,p)$.
O. B. d. A. sei $k = (p_1, p_2, \ldots, p_n)$.
Gegeben ist eine bei p_{n-1} **vorwärts**gültige Formel Φ, wobei p_{n-1} bezüglich Φ nicht abblockend.
Also gilt $\alpha)$.

zu β) Sei $P_1^2(\mathrm{end}(B)) = j$.

Fallunterscheidung:

β1) α(i) ist verallgemeinerte Wertzuweisung:

Analog zu ii) β) gilt $\mathrm{vtraf}((i,j),\Phi)^{J',P_2^2(\mathrm{end}(B))} = W$.

Also mit $\Phi \in \{\Phi_k \mid (k,j) \in R\}$ gilt

$$\Big(\bigvee_{k \in \{i' \mid (i',j) \in R\}} \mathrm{vtraf}((k,j),\Phi_k)\Big)^{J',P_2^2(\mathrm{end}(B))} = W.$$

β2) α(i) ist keine verallgemeinerte Wertzuweisung.

Analog zu ii2) β) gilt:

$$\mathrm{vtraf}((i,j),\Phi)^{J',P_2^2(\mathrm{end}(B))} = W.$$

<u>Bemerkung</u>

Regel a) von Satz 3 ermöglicht insbesondere die notationelle Vereinfachung vorwärtsgültiger Formeln:
z. B. ist die Formel (*)

$$(\exists\, v'_{11},\ldots,v'_{mn_m})(\Phi(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}}) \wedge t_1(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}}) = (v_{11},\ldots,v_{1n_1}) \wedge$$

$$\wedge \ldots \wedge t_m(^{v_{11}\ldots v_{mn_m}}_{v'_{11}\ldots v'_{mn_m}}) = (v_{m1},\ldots,v_{mn_m}))$$

unter der Voraussetzung, daß $v_{11},\ldots,v_{mn_m}$ weder in Φ noch in $t_1,t_2,\ldots,t_m$ vorkommen, zu der Formel

$$\Phi \wedge t_1 = (v_{11},\ldots,v_{1n_1}) \wedge \ldots \wedge t_m = (v_{m1},\ldots,v_{mn_m})$$

logisch äquivalent.

Ein weiterer Spezialfall zur Vereinfachung vorwärtsgültiger Formeln liegt vor, wenn die $v_{11},\ldots,v_{mn_m}$ in Φ frei vorkommen und Φ bezüglich J' in ein äquivalentes Φ' umgeformt werden kann für das gilt:

$(\forall\ 1\leq i\leq m)\, v_{i1},\ldots,v_{in_i}$ kommen in Φ nur noch in Form des Terms t_i vor.

In diesem Fall ist die Formel Φ'', die man erhält, wenn in Φ' für $1\leq i\leq m$ jeder Term t_i durch $v_{i1},\ldots,v_{in_i}$ ersetzt wird, eine zu (*) äquivalente, jedoch notationell sehr viel einfachere Formel.

Mischformen obiger ausgezeichneter Spezialfälle führen zu weiteren Vereinfachungs-Regeln.

<u>Def. 4</u>

Seien $\Phi \in \mathrm{Form}(A')$ und $(i,j) \in R$ beliebig.

Die <u>Rückwärtstransformation von Φ bezüglich (i,j)</u> (kurz $\mathrm{rtraf}((i,j),\Phi)$ ist definiert durch:

i) Falls $\alpha(i)$ ein Modulkopf, eine Zweigmarkierung, eine Parallelverzweigung, ein Einfachzweig-Abschluß, ein Schleifenrücksprung, ein Schleifeneingang, ein Einfachzweig-Sammelpunkt oder eine einfache Zusicherung, so gilt:

$$\mathrm{rtraf}((i,j),\Phi)) \leftrightharpoons \begin{cases} \Phi', & \text{falls in } \alpha(j) \text{ eine Zeichenkette } \{\Phi'\} \text{ mit } \Phi' \in \mathrm{Form}(A') \text{ vorkommt} \\ \Phi & \text{sonst.} \end{cases}$$

ii) Falls $\alpha(i)$ eine verallgemeinerte Wertzuweisung und

$$((t_1 \rightleftharpoons (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \rightleftharpoons (v_{m1},\ldots,v_{mn_m}))$$

die in $\alpha(i)$ vorkommende Wertzuweisung (mit $(\forall\ 1\leq k\leq m)\ t_k \neq$ <u>frei</u>), so gilt:

$$\mathrm{rtraf}((i,j),\Phi) \leftrightharpoons \psi\left(\begin{matrix} v_{11} & \ldots v_{1n_1} & \ldots v_{m1} & \ldots v_{mn_m} \\ P_1^{n_1}(t_1) \ldots P_{n_1}^{n_1}(t_1) & \ldots P_1^{n_m}(t_m) & \ldots P_{n_m}^{n_m}(t_m) & \end{matrix}\right)$$

$$\text{wobei } \psi \leftrightharpoons \begin{cases} \Phi', & \text{falls in } \alpha(j) \text{ eine Zeichenkette } \{\Phi'\} \text{ mit } \Phi' \in \mathrm{Form}(A') \text{ vorkommt} \\ \Phi & \text{sonst.} \end{cases}$$

iii) Falls $\alpha(i)$ eine dezentrale Verzweigung der Form

<u>if</u> B,

so gilt:

$$\mathrm{rtraf}((i,j),\Phi) \leftrightharpoons \begin{cases} (B \rightarrow \Phi), & \text{falls } j = i+1 \\ (\neg B \rightarrow \Phi) & \text{sonst.} \end{cases}$$

iv) Falls $\alpha(i)$ eine zentrale Verzweigung der Form

<u>case</u> $(B_1, B_2, \ldots, B_n)$,

so gilt:

$$\mathrm{rtraf}((i,j),\Phi) \leftrightharpoons \begin{cases} (B_1 \rightarrow \Phi), & \text{falls } j = i+1 \\ (\neg B_1 \wedge B_2 \rightarrow \Phi), & \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = 1 \\ \vdots & \\ (\neg B_1 \wedge \ldots \wedge \neg B_{n-1} \wedge B_n \rightarrow \Phi), & \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = n-1 \\ (\neg B_1 \wedge \ldots \wedge \neg B_n \rightarrow \Phi) & \text{sonst.} \end{cases}$$

v) Falls $\alpha(i)$ eine zentrale Verzweigung der Form

<u>case</u> <u>one</u> $(B_1, B_2, \ldots, B_n)$,

so gilt:

$$\mathrm{rtraf}((i,j),\Phi) \leftrightharpoons \begin{cases} (B_1' \rightarrow \Phi), & \text{falls } j = i+1 \\ (B_2' \rightarrow \Phi), & \text{falls } |\{j' \mid (i,j') \in R \wedge j' < j\}| = 1 \\ \vdots & \\ (B_n' \rightarrow \Phi), & \text{falls } |\{j' \mid (i,j') \in R \wedge j'\ j\}| = n-1 \\ (\neg B_1 \wedge \neg B_2 \wedge \ldots \wedge \neg B_n \rightarrow \Phi) & \text{sonst,} \end{cases}$$

wobei für alle $1 \leq k \leq n$:

$$B_k' = \neg B_1 \wedge \ldots \wedge \neg B_{k-1} \wedge B_k \wedge \neg B_{k+1} \wedge \ldots \wedge \neg B_n.$$

Satz 5

(Kalkül zur Entwicklung rückwärts-gültiger Formeln)

Sei $j \in P$ beliebig und $\Phi \in Form(A')$ eine beliebige beim Punkt j rückwärtsgültige Formel.

a) Für alle $\Phi' \in Form(A')$ mit

$$(\forall\, \sigma \in \Sigma_J)(\Phi'^{J',\sigma} = W \rightarrow \Phi^{J',\sigma} = W)$$

gilt: Φ' ist rückwärtsgültig bei j.

b) Sei $i \in P$ beliebig mit $(i,j) \in R$.

b1) Ist $\{j' \mid (i,j') \in R\} = \{j\}$[1] und gilt: $(i, rtraf((i,j),\Phi)) \nvdash \Omega$, so ist $rtraf((i,j),\Phi)$ rückwärtsgültig bei i.

b2) Ist $\{j' \mid (i,j') \in R\} \supset \{j\}$ und ist für alle $k \in \{j' \mid (i,j') \in R\}$ Φ_k eine bei k rückwärtsgültige Formel mit $(i, rtraf((i,k),\Phi_k)) \nvdash \Omega$, so gilt:

$$\bigvee_{k \in \{j' \mid (i,j') \in R\}} rtraf((i,k),\Phi_k)$$

ist rückwärtsgültig bei i.

Beweis

a) Sei $\sigma \in \Sigma_J$ beliebig mit $\Phi'^{J',\sigma} = W$.

Da Φ rückwärtsgültig bei j und $\Phi^{J',\sigma} = W$, gilt:

$$\exists\, B\ (B \text{ ist direkte Berechnung auf } M \wedge anf(B) = (j,\sigma) \wedge P_1^2(end(B)) \in ZN(j) \wedge \psi(P_1^2(end(B)))^{J',P_2^2(end(B))} = W,$$

d. h. Φ' ist rückwärtsgültig bei j.

---

1) D. h. $\alpha(i)$ ist keine zentrale oder dezentrale Verzweigung.

b1) Sei $\sigma \in \Sigma_J$ beliebig mit $\mathrm{rtraf}((i,j),\Phi)^{J',\sigma} = W$.

Zu zeigen:

$\exists$ B (B ist direkte Berechnung auf M $\wedge$

$\mathrm{anf}(B) = (i,\sigma) \wedge P_1^2(\mathrm{end}(B)) \in ZN(i) \wedge$

$\psi(P_1^2(\mathrm{end}(B)))^{J',P_2^2(\mathrm{end}(B))} = W$).

Fallunterscheidung:

I) $j \notin ZS$, d. h. $ZN(i) = ZN(j)$.

I1) $\alpha(i)$ ist keine verallgemeinerte Wertzuweisung (und nach Voraussetzung auch keine zentrale oder dezentrale Verzweigung), d. h. es gilt:

$\mathrm{rtraf}((i,j),\Phi) = \Phi$, $(i,\Phi) \not\mapsto \Omega$ und

$\mathrm{modfu}(i,\sigma) = (j,\sigma)$. (*)

Da Φ rückwärtsgültig bei j, gilt mit (*) die Behauptung.

I2) $\alpha(i)$ ist verallgemeinerte Wertzuweisung, wobei o.B.d.A. $((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$ die in $\alpha(i)$ vorkommende Wertzuweisung sei (mit $(\forall\ 1 \leq k \leq m)\ t_k \neq \underline{\text{frei}}$).

Wissen: $\mathrm{modfu}(i,\sigma) = (j,\sigma')$ (*) mit

$$\sigma' = \left\langle x \mapsto \begin{cases} P_1^{n_1}(t_1^{J,\sigma}), & \text{falls } x = v_{11} \\ \vdots & \\ P_{n_m}^{n_m}(t_m^{J,\sigma}), & \text{falls } x = v_{mn_m} \\ \sigma(x) & \text{sonst} \end{cases} \right\rangle$$

und mit $\mathrm{rtraf}((i,j),\Phi) = \Phi\left(\begin{matrix} v_{11} & \ldots v_{mn_m} \\ P_1^{n_1}(t_1) & \ldots P_{n_m}^{n_m}(t_m) \end{matrix}\right)$

gilt: $\Phi^{J',\sigma'} = W$.

Da Φ rückwärtsgültig bei j, gilt mit (*) die Behauptung.

II) $j \in ZS$, d. h. $ZN(i) = \{j\}$.

Da nach Voraussetzung $(i, rtraf((i,j),\Phi)) \nvdash \Omega$,

gilt: $(i,\sigma) \nvdash \Omega$ und mit $\{j' \mid (i,j') \in R\} = \{j\}$

gilt: $P_1^2(modfu(i,\sigma)) = j$.

Zu zeigen bleibt: $\psi(j)^{J',P_2^2(modfu(i,\sigma))} = W$.

II1) $\alpha(i)$ ist keine verallgemeinerte Wertzuweisung,

d. h. es gilt $rtraf((i,j),\Phi) = \psi(j) \wedge$

$$P_2^2(modfu(i,\sigma)) = \sigma.$$

Nach Voraussetzung gilt somit:

$$\psi(j)^{J',P_2^2(modfu(i,\sigma))} = W.$$

II2) $\alpha(i)$ ist verallgemeinerte Wertzuweisung.

Entsprechend II1), analog zu I2).

b2) Sei $\sigma \in \Sigma_J$ beliebig mit

$$\left(\bigvee_{k \in \{j' \mid (i,j') \in R\}} rtraf((i,k),\Phi_k) \right)^{J',\sigma} = W.$$

Zu zeigen:

$\exists$ B (B ist direkte Berechnung auf M $\wedge$ anf(B) = (i,σ)

$\wedge\ P_1^2(end(B)) \in ZN(i) \wedge \psi(P_1^2(end(B)))^{J',P_2^2(end(B))} = W)$).

O.B.d.A. gelte: für $k \in \{j' \mid (i,j') \in R\}$

$$rtraf((i,k),\Phi_k)^{J',\sigma} = W.$$

Wissen: $\alpha(i)$ ist zentrale oder dezentrale Verzweigung und $k \in ZS$. (Def. der Syntax)

Zu zeigen bleibt:

$$(i,\sigma) \nvdash \Omega \wedge modfu(i,\sigma) = (k,\sigma).$$

Fallunterscheidung:

I) $\alpha(i)$ ist dezentrale Verzweigung der Form <u>if</u> B:

I1) $\mathrm{rtraf}((i,k),\Phi_k) = (B \rightarrow \Phi_k)$, d. h. $k = j+1$.

Nach Voraussetzung gilt: $(B \rightarrow \Phi_k)^{J',\sigma} = W$,

d. h. $B^{J',\sigma} \neq \Omega$, also $(i,\sigma) \not\vdash \Omega$.

Somit gilt $\mathrm{modfu}(i,\sigma) = (k,\sigma)$.

I2) $\mathrm{rtraf}((i,k),\Phi_k) = (\neg B \rightarrow \Phi_k)$, d. h. $k \neq j+1$.

Analog I1).

II) $\alpha(i)$ ist dezentrale Verzweigung.
Analog I).

Def. 6

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$.

Ein Kantenzug $kz \in K(j,k)$ heißt partiell korrekt $\leftrightharpoons$

$\forall$ B (B ist Berechnung auf M $\wedge$ $pf(B) = anf(kz,P_1^2(end(B)))$

$\wedge\ \psi(j)^{J',P_2^2(anf(B))} = W \rightarrow$

$(P_1^2(end(B)) \neq k \rightarrow end(B) \nvdash \Omega) \wedge$

$(P_1^2(end(B)) = k \rightarrow \psi(k)^{J',P_2^2(end(B))} = W))$.

Satz 7

M ist partiell korrekt gdw. gilt:

$(\forall\ j,k \in ZS)$ $(\forall\ kz \in K(j,k))$ kz ist partiell korrekt.

Beweis

" $\Rightarrow$ " Siehe Def. 1 (3.1.2.1.1)

" $\Leftarrow$ " Mit Lemma 4 (3.1.2.1.1)

Def. 8

Seien $j,k \in ZS$ beliebig mit $K(j,k) \neq \emptyset$, $kz = (p_o,p_1,\ldots,p_q) \in K(j,k)$,

$\Phi \in Form(A')$ und i mit $0 \leq i \leq q$ beliebig.

a) Φ ist bezüglich kz vorwärtsgültig an Stelle i $\leftrightharpoons$

$(i = 0 \rightarrow (\psi(j) \Rightarrow \Phi)^{1)}) \wedge (i>0 \rightarrow$

$\forall$ B (B ist Berechnung auf M $\wedge$ $\psi(j)^{J',P_2^2(anf(B))} = W \rightarrow$

$(\forall\ p \in P(kz)\setminus\{p_i\})(anf(kz,p) \neq \emptyset \wedge anf(kz,p) = pf(B,p)$

$\rightarrow z(B,p) \nvdash \Omega)^{2)} \wedge$

$(pf(B) = anf(kz,p_i) \rightarrow \Phi^{J',P_2^2(end(B))} = W)))$.

---

1) Es gelte: $(\Phi \Rightarrow \Phi') \leftrightharpoons (\forall\ \sigma \in \Sigma_J)(\Phi^{J',\sigma} = W \rightarrow \Phi'^{J',\sigma} = W)$.

2) Für $p = p_o$ gilt mit $p_o \in ZS$ stets: $z(B,p_o) \nvdash \Omega$.

b) <u>Φ ist bezüglich kz rückwärtsgültig an Stelle i</u> $\leftrightharpoons$

$(i = q \rightarrow (\Phi \Rightarrow \psi(k))) \wedge (i<q \rightarrow$

$(\forall \sigma \in \Sigma_J)(\Phi^{J',\sigma} = W \rightarrow \exists B$ (B ist direkte Berechnung auf M $\wedge$

$\mathrm{anf}(B) = (p_i,\sigma) \wedge \mathrm{pf}(B) = \mathrm{end}(kz,p_i) \wedge$

$\psi(k)^{J',P_2^2(\mathrm{end}(B))} = W)))$.

<u>Satz 9</u>

(Kalkül zur Entwicklung von bezüglich Kantenzügen vorwärts-gültigen Formeln)

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$, $kz = (p_o,p_1,\ldots,p_q) \in K(j,k)$, $\Phi \in \mathrm{Form}(A')$ und i mit $0 \leq i \leq q$ beliebig.
Φ sei bezüglich kz vorwärtsgültig an Stelle i.

a) Für alle $\Phi' \in \mathrm{Form}(A')$ mit $(\Phi \Rightarrow \Phi')$ gilt:
Φ' ist bezüglich kz vorwärtsgültig an Stelle i.

b) Sei $i<q$.
Ist p_i bezüglich Φ nicht abblockend, so gilt:
$\mathrm{vtraf}((p_i,p_{i+1}),\Phi)$ ist bezüglich kz vorwärtsgültig an Stelle i+1.

<u>Beweis</u>

Weitgehend analog zu Satz 16 bzw. Satz 3.

<u>Satz 10</u>

(Vorwärts-Entwicklungs-Methode zum Nachweis der Korrektheit von Kantenzügen)[1)]

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$ und $kz = (p_o,p_1,\ldots,p_q) \in K(j,k)$ beliebig.

---

1) Diese Methode wurde informell von Floyd [1967] eingeführt.

Existiert eine Folge $\Phi_o, \Phi_1, \ldots, \Phi_q$ mit der Eigenschaft

i) $\psi(j) \Rightarrow \Phi_o$ und

ii) $(\forall\ 0 \leq i < q)((p_i, \Phi_i) \not\vdash \Omega \wedge (\text{vtraf}((p_i, p_{i+1}), \Phi_i) \Rightarrow \Phi_{i+1}))$ und

iii) $\Phi_q \Rightarrow \psi(k)$,

so ist kz partiell korrekt.

<u>Beweis</u>

<u>Hilfssatz:</u>

$(\forall\ 0 \leq i \leq q)(\Phi_i$ ist bezüglich kz vorwärtsgültig an Stelle i).

<u>Beweis</u>

Induktionsanfang:

Da $\psi(j)$ bezüglich kz vorwärtsgültig an Stelle 0 gilt mit $\psi(j) \Rightarrow \Phi_o$ nach Satz 9a), daß Φ_o bezüglich kz vorwärtsgültig an Stelle 0.

Induktionsschritt:

Sei Φ_i bezüglich kz vorwärtsgültig an Stelle i mit $i<q$.

Nach Voraussetzung gilt $(p_i, \Phi_i) \not\vdash \Omega$.

Nach Satz 9b) ist somit $\text{vtraf}((p_i, p_{i+1}), \Phi_i)$ bzgl. kz vorwärtsgültig an Stelle i+1.

Nach Voraussetzung gilt weiter $\text{vtraf}((p_i, p_{i+1}), \Phi_i) \Rightarrow \Phi_{i+1}$.

Somit ist nach Satz 9a) Φ_{i+1} bezüglich kz vorwärtsgültig an Stelle i+1.

Da Φ_q bezüglich kz vorwärtsgültig an Stellt q, $p_q = k$ und mit $\Phi_q \Rightarrow \psi(k)$ $\psi(k)$ bezüglich kz vorwärtsgültig an Stelle k, gilt die Behauptung.

<u>Bemerkung</u>

Für $\Phi_{i+1} = \text{vtraf}((p_i, p_{i+1}), \Phi_i)$ (für alle $0 \leq i < q$) und $\Phi_0 = \psi(j)$ reduziert sich der Nachweis der Korrektheit von kz - sieht man vom Nachweis des Nicht-Abblockens $((p_i, \Phi_i) \not\vdash \Omega)$ ab - auf den Beweis der "Verifikations-Bedingung" $\Phi_q \Rightarrow \psi(k)$. Jedes Φ_i ($1 \leq i \leq q$ beliebig) ist hierbei die <u>stärkste Nachbedingung</u> (strongest post condition) bezüglich $\psi(j)$ bzw. Φ_{i-1}.

Satz 11

(Kalkül zur Entwicklung von bezüglich Kantenzügen rückwärts-gültigen Formeln)

Seien $j,k \in ZS$ beliebig mit $K(j,k) \neq \emptyset$, $kz = (p_0,p_1,\ldots,p_q) \in K(j,k)$, $\Phi \in Form(A')$ und $0 \leq i \leq q$ beliebig.
Φ sei bezüglich kz rückwärtsgültig an Stelle i.

a) Für alle $\Phi' \in Form(A')$ mit $(\Phi' \Rightarrow \Phi)$ gilt:
Φ' ist bezüglich kz rückwärtsgültig an Stelle i.

b) Sei $i>0$.
Gilt $(p_{i-1}, rtraf((p_{i-1},p_i),\Phi)) \nvdash \Omega$, so ist $rtraf((p_{i-1},p_i),\Phi)$ bezüglich kz rückwärtsgültig an Stelle i-1.

Beweis

Weitgehend analog zu Satz 16 bzw. Satz 5.

Satz 12

(Rückwärts-Entwicklungs-Methode zum Nachweis der Korrektheit von Kantenzügen)

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$ und $kz = (p_0,p_1,\ldots,p_q) \in K(j,k)$ beliebig.

Existiert eine Folge $\Phi_q,\Phi_{q-1},\ldots,\Phi_0$ mit der Eigenschaft

i) $\Phi_q \Rightarrow \psi(k) \wedge$

ii) $(\forall\ 0<i\leq p)((p_{i-1}, rtraf((p_{i-1},p_i),\Phi_i)) \nvdash \Omega \wedge$
$(\Phi_{i-1} \Rightarrow rtraf((p_{i-1},p_i),\Phi_i))) \wedge$

iii) $\psi(j) \Rightarrow \Phi_0$,

so ist kz partiell korrekt.

Beweis

Hilfssatz:

$(\forall\ 0\leq i\leq q)(\Phi_i$ ist bezüglich kz rückwärtsgültig an Stelle i).

<u>Beweis</u>

Induktionsanfang:

Mit $\Phi_q \Rightarrow \psi(k)$ gilt: Φ_q ist bezüglich kz rückwärtsgültig an Stelle q.

Induktionsschritt:

Sei Φ_i bezüglich kz rückwärtsgültig an Stelle i mit i>0.

Nach Voraussetzung gilt $(p_{i-1}, rtraf((p_{i-1}, p_i)\Phi_i)) \not\vdash \Omega$

und somit nach Satz 11b):

$rtraf((p_{i-1}, p_i), \Phi_i)$ ist bezüglich kz rückwärtsgültig an Stelle i-1.

Mit $(\Phi_{i-1} \Rightarrow rtraf((p_{i-1}, p_i), \Phi_i))$ gilt nach Satz 11a):

Φ_{i-1} ist bezüglich kz rückwärtsgültig an Stelle i-1.

Sei nun B eine beliebige Berechnung auf M mit

$pf(B) = anf(kz, P_1^2(end(B)))$ und $\psi(j)^{J', P_2^2(anf(B))} = W$.

Da nach Voraussetzung $(\psi(j) \Rightarrow \Phi_o)$ gilt und Φ_o bezüglich kz rückwärtsgültig an Stelle 0, ist nach Satz 11a) $\psi(j)$ bezüglich kz rückwärtsgültig an Stelle 0.

Somit gilt:

$\exists$ B' (B' ist direkte Berechnung auf M $\wedge$

$$anf(B') = (p_o, P_2^2(anf(B))) \wedge pf(B') = end\ (kz, p_o)$$

$$\wedge\ \psi(k)^{J', P_2^2(end(B'))} = W).$$

Da anf(B') = anf(B) gilt B' = B (d. h. $P_1^2(end(B)) = k$) oder B ist Anfangsstück der Berechnung B' (d. h. $P_1^2(end(B)) \neq k$).

1. Fall: $P_1^2(end(B)) = k$.

   Mit B' = B gilt end(B') = end(B) und somit gilt

   $$\psi(k)^{J', P_2^2(end(B))} = W.$$

2. Fall: $P_1^2(end(B)) \neq k$.

   Da B Anfangsstück von B', besitzt end(B) eine Folgekonfiguration, d. h. es gilt end(B) $\not\vdash \Omega$.

Bemerkung

Da die Rückwärtstransformation von Formeln zu einfacher handhabbaren Ausdrücken führt als die Vorwärtstransformation, wird in Verifikationssystemen (siehe z. B. King [1969], Waldinger/Levitt [1974]) die Rückwärtstransformation bevorzugt.
Die Frage des Nichtabblockens wird in diesen Systemen jedoch noch weitgehend ausgeklammert.
Analog zur Vorwärts-Entwicklungs-Methode kann auch bei der Rückwärts-Entwicklungs-Methode der Beweis der Korrektheit eines Kantenzugs weitgehend auf den Beweis einer "Verifikations-Bedingung" $\psi(j) \Rightarrow \Phi_o$ reduziert werden. Die Φ_{i-1}, für die dann gilt $\Phi_{i-1} = \text{rtraf}((p_{i-1},p_i),\Phi_i)$, nennt man dann jeweils schwächste Vorbedingung (weakest precondition) bezüglich $\psi(k)$ bzw. Φ_i, weshalb z. B. Basu/Yeh [1975] und Dijkstra [1975] auch von der Methode der schwächsten Vorbedingung sprechen.

Def. 13

a) Ein symbolischer Ausdruck ist ein Paar(Φ,S) mit $\Phi \in \text{Form}(A')$ und $S|\text{Var}(M) \to \{t|t \in \text{Term}(A') \wedge \text{wst}(t) = 1\}$.

b) Seien (Φ,S) und (Φ',S') zwei symbolische Ausdrücke.
Mit (Φ,S) gilt auch (Φ',S') (kurz $((\Phi,S) \Rightarrow (\Phi',S'))$) $\leftrightharpoons$ $(\Phi \Rightarrow \Phi') \wedge (\forall\, v \in \text{Var}(M))\ S(v) \equiv S'(v)$ [1].

Def. 14

Sei (Φ,S) ein beliebiger symbolischer Ausdruck und $(i,j) \in R$ beliebig.
Die symbolische Vorwärtstransformation von (Φ,S) bezüglich (i,j) (kurz svtraf$((i,j),(\Phi,S))$) ist definiert durch:

---

1) Für alle $t,t' \in \text{Term}(A')$ gelte:
$t \equiv t' \;\leftrightharpoons\; (\forall\, v \in \Sigma_J)\ t^{J',\sigma} = t'^{J',\sigma}$.

i) Falls $\alpha(i)$ eine verallgemeinerte Wertzuweisung der Form
$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$,
so gilt:
$\mathrm{svtraf}((i,j),(\Phi,S)) \leftrightharpoons (\Phi,S')$ mit

$$S' \leftrightharpoons \left\langle v \mapsto \begin{cases} P_1^{n_1}(t_1(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})), & \text{falls } v = v_{11} \\ \vdots & \vdots \\ P_{n_1}^{n_1}(t_1(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})), & \text{falls } v = v_{1n_1} \\ P_1^{n_2}(t_2(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})), & \text{falls } v = v_{21} \\ \vdots & \vdots \\ P_{n_m}^{n_m}(t_m(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})), & \text{falls } v = v_{mn_m} \\ S(v) & \text{sonst} \end{cases} \mid v \in \mathrm{Var}(M) \right\rangle$$

ii) Falls $\alpha(i)$ keine verallgemeinerte Wertzuweisung, so gilt
$\mathrm{svtraf}((i,j),(\Phi,S)) \leftrightharpoons (\Phi',S)$ mit

α) falls $\alpha(i)$ eine dezentrale Verzweigung der Form

<u>if</u> B

so gilt:

$$\Phi' \leftrightharpoons \begin{cases} (\Phi \wedge B(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})), & \text{falls } j = i+1 \\ (\Phi \wedge \neg B(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})) & \text{sonst} \end{cases}$$

(d. h. $\Phi' \leftrightharpoons \mathrm{vtraf}((i,j),\Phi)$, wenn sich $\mathrm{vtraf}((i,j),\Phi)$ auf einen Modul M bezieht, der sich von M' genau dadurch unterscheidet, daß anstelle von $\alpha(i)$ in M' die elementare Moduloperation <u>if</u> B' mit $B' = B(^{v_1 \ldots v_m}_{S(v_1)\ldots S(v_m)})$ steht)

β) falls $\alpha(i)$ eine zentrale Verzweigung der Form

<u>case</u> mod $(B_1, B_2, \ldots, B_n)$

so gilt:

$\Phi' \leftrightharpoons \mathrm{vtraf}((i,j),\Phi)$,

wobei sich $\mathrm{vtraf}((i,j),\Phi)$ auf einen Modul M' beziehe, der sich von M genau dadurch unterscheidet, daß anstelle von $\alpha(i)$ in M' die elementare Moduloperation

<u>case</u> mod $(B'_1, \ldots, B'_n)$

steht mit $B'_i \leftrightharpoons B_i\left(\begin{smallmatrix} v_1 & \ldots v_m \\ S(v_1) & \ldots S(v_m) \end{smallmatrix}\right)$ für $1 \leq i \leq n$.

γ) $\Phi' \leftrightharpoons \Phi$, falls $\alpha(i)$ keine zentrale oder dezentrale Verzweigung.

<u>Def. 15</u>

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$,
$kz = (p_o, p_1, \ldots, p_q) \in K(j,k)$, (Φ,S) ein symbolischer Ausdruck und i mit $0 \leq i \leq q$ beliebig.

<u>(Φ,S) ist bezüglich kz vorwärtsgültig an Stelle i</u> $\leftrightharpoons$

$(i = 0 \rightarrow (\psi(j) \Rightarrow \Phi) \wedge S = \mathrm{Id}_{\mathrm{Var}(M)}$ [1]$) \wedge (i>0 \rightarrow$

$\forall B\ (B$ ist Berechnung auf $M \wedge \psi(j)^{J',P_2^2(\mathrm{anf}(B))} = W \rightarrow$

$(\forall p \in P(kz) \setminus \{p_i\})(\mathrm{anf}(kz,p) \neq \emptyset \wedge \mathrm{anf}(kz,p) = \mathrm{pf}(B,p)$
$\rightarrow z(B,p) \not\mapsto \Omega) \wedge$

$(\mathrm{pf}(B) = \mathrm{anf}(kz,p_i) \rightarrow (\Phi^{J',P_2^2(\mathrm{anf}(B))} = W \wedge$

$(\forall v \in \mathrm{Var}(M))\ P_2^2(\mathrm{end}(B))(v) =$

$(S(v))^{J',P_2^2(\mathrm{anf}(B))}))))$.

---

1) S ist die identische Abbildung auf Var(M).

Satz 16

(Kalkül zur Entwicklung von bezüglich Kantenzügen vorwärtsgültigen symbolischen Ausdrücken)

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$,
$kz = (p_o,p_1,\ldots,p_q) \in K(j,k)$, (Φ,S) ein symbolischer Ausdruck und i mit $0 \leq i \leq q$ beliebig.
(Φ,S) sei bezüglich kz vorwärtsgültig an Stelle i.

a) Für alle symbolischen Ausdrücke (Φ',S') mit $(\Phi,S) \Rightarrow (\Phi',S)$ gilt: (Φ',S') ist bezüglich kz vorwärtsgültig an Stelle i.

b) Sei $i<q$.

Wenn $(\forall\, v \in \Sigma_{J'})(\Phi^{J',\sigma} = W \rightarrow (p_i,\sigma(\begin{smallmatrix} v_1 & \ldots v_m \\ S(v_1) & \ldots S(v_m) \end{smallmatrix})) \not\vdash \Omega)$,

so ist $svtraf((p_i,p_{i+1}),(\Phi,S))$ bezüglich kz vorwärtsgültig an Stelle i+1.

Beweis

Zu a)

Sei (Φ',S') ein symbolischer Ausdruck mit
$(\Phi \Rightarrow \Phi') \wedge (\forall\, v \in Var(M))\ S(v) \equiv S'(v)$.

α) $i = 0$:
Mit $(\psi(j) \Rightarrow \Phi)$ und $(\Phi \Rightarrow \Phi')$ gilt
$(\psi(j) \Rightarrow \Phi')$.

β) $i>0$:

Sei B beliebige Berechnung auf M mit

$$\psi(j)^{J',P_2^2(anf(B))} = W.$$

Zu zeigen bleibt:

$$(pf(B) = anf(kz,p_i) \rightarrow (\Phi'^{J',P_2^2(anf(B))} = W \wedge$$

$$(\forall\, v \in Var(M))\ P_2^2(end(B))(v) = (S'(v))^{J',P_2^2(anf(B))}).$$

Sei $pf(B) = anf(kz,p_i)$.

Mit $\Phi^{J',P_2^2(anf(B))} = W$ und $(\Phi \Rightarrow \Phi')$

gilt $\Phi'^{J',P_2^2(anf(B))} = W$.

Sei $v \in Var(M)$ beliebig.

Mit $S(v) \equiv S'(v)$ und $P_1^2(end(B))\ (v) = S(v))^{J',P_2^2(anf(B))}$

gilt $P_2^2(end(B))\ (v) = (S'(v))^{J',P_2^2(anf(B))}$.

Zu b)

Es gilt: $0<i+1\leq q$.

Sei B beliebige Berechnung auf M mit

$\psi(j)^{J',P_2^2(anf(B))} = W$.

Zu Zeigen:

α) Sei $p \in P(kz)\setminus\{p_{i+1}\}$ beliebig mit

$anf(kz,p) \neq \emptyset$ und $anf(kz,p) = pf(B,p)$.

Zu zeigen: $z(B,p) \not\vdash \Omega$.

Fallunterscheidung:

α1) $p \neq p_i$: Da (Φ,S) bezüglich kz vorwärtsgültig an p_i gilt $z(B,p) \not\vdash \Omega$.

α2) $p = p_i$:

Nach Voraussetzung gilt:

$\Phi^{J',P_2^2(anf(B))} = W$ und

$P_2^2(z(B,p_i))\ (v) = (S(v))^{J',P_2^2(anf(B))}$ für $v \in Var(M)$ beliebig.

Somit gilt:

$(p_i, \sigma(\begin{smallmatrix} v_1 & \ldots v_m \\ S(v_1) & \ldots S(v_m) \end{smallmatrix})) = z(B,p_i)$ und mit

$(p_i, \sigma(\begin{smallmatrix} v_1 & \ldots v_m \\ S(v_1) & \ldots S(v_m) \end{smallmatrix})) \not\mapsto \Omega$ gilt: $z(B,p_i) \not\mapsto \Omega$.

β) Sei $pf(B) = anf(kz,p_{i+1})$ und $svtraf((p_i,p_{i+1}),(\Phi,S)) = (\Phi',S')$.

β1) $\alpha(i)$ ist verallgemeinerte Wertzuweisung der Form
$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_{m'} \Rightarrow (v_{m'1},\ldots,v_{m'n_{m'}})))$.

Da $\Phi = \Phi'$, bleibt zu zeigen:

$(\forall\, v \in Var(M))\ P_2^2(end(B))\ (v) = (S'(v))^{J',P_2^2(anf(B))}$.

Sei $v \in Var(M)$ beliebig.

i) $v \notin \{v_{11},\ldots,v_{m'n_{m'}}\}$.

Es gilt: $S'(v) = S(v)$.
Nach Voraussetzung gilt:

$P_2^2(z(B,p_i))(v) = (S(v))^{J',P_2^2(anf(B))}$.

Mit $modfu(z(B,p_i)) = end(B)$ und

$P_2^2(end(B))(v) = P_2^2(z(B,p_i))(v)$ gilt somit:

$P_2^2(end(B))(v) = (S'(v))^{J',P_2^2(anf(B))}$.

ii) $v \in \{v_{11},\ldots,v_{m'n_{m'}}\}$.

O.B.d.A. sei $v = v_{11}$.

Es gilt: $S'(v) = P_1^{n_1}(t_1(\begin{smallmatrix} v_1 & \ldots v_m \\ S(v_1) & \ldots S(v_m) \end{smallmatrix}))$.

Nach Voraussetzung gilt:

$P_2^2(z(B,p_i))(v) = (S(v))^{J',P_2^2(anf(B))}$.

Mit $modfu(z(B_i,p_i)) = end(B)$ und

$P_2^2(end(B))(v) = P_1^{n_1}(t_1^{J',P_2^2(z(B,p_i))})$ gilt somit:

$$P_2^2(\mathrm{end}(B))(v) = P_1^{n_1}((t_1(^{v_1 \quad \ldots v_m}_{S(v_1)\ldots S(v_m)}))^{J',P_2^2(\mathrm{anf}(B))}$$

$$= S'(v)^{J',P_2^2(\mathrm{anf}(B))} .$$

β2) α(i) ist keine verallgemeinerte Wertzuweisung.
Wir beschränken uns auf den Fall, daß α(i) eine dezentrale Verzweigung der Form

<u>if</u> G und $p_{i+1} = p_i+1$.

(Die anderen Fälle beweist man analog.)

Da $S = S'$, $\Phi' = (\Phi \wedge G(^{v_1 \quad \ldots v_m}_{S(v_1)\ldots S(v_m)}))$ und

$\Phi^{J',P_2^2(\mathrm{anf}(B))} = W$, bleibt zu zeigen:

$(G(^{v_1 \quad \ldots v_m}_{S(v_1)\ldots S(v_m)}))^{J',P_2^2(\mathrm{anf}(B))} = W.$

Da nach Voraussetzung mit $\mathrm{modfu}(z(B,p_i)) = \mathrm{end}(B)$ gilt:

$(\forall\ v \in \mathrm{Var}(M))\ P_2^2(\mathrm{end}(B))(v) = S(v))^{J',P_2^2(\mathrm{anf}(B))}$ und

$G^{J',P_2^2(\mathrm{end}(B))} = W$ gilt:

$G(^{v_1 \quad \ldots v_m}_{S(v_1)\ldots S(v_m)})^{J',P_2^2(\mathrm{anf}(B))} = W.$

<u>Satz 17</u>

(Symbolische Vorwärts-Entwicklungs-Methode zum Nachweis der Korrektheit von Kantenzügen)

Seien $j,k \in ZS$ mit $K(j,k) \neq \emptyset$ und
$kz = (p_o,p_1,\ldots,p_q) \in K(j,k)$ beliebig.
Existiert eine Folge $(\Phi_o,S_o),(\Phi_1,S_1),\ldots,(\Phi_q,S_q)$ symbolischer Ausdrücke mit der Eigenschaft

i) $(\psi(j),\mathrm{Id}_{\mathrm{Var}(M)}) \Rightarrow (\Phi_o,S_o)$ und

ii) $(\forall\ 0 \leq i < q)((\forall\ \sigma \in \Sigma_J)(\Phi_i^{J',\sigma} = W \rightarrow (p_i, \sigma(S_i^{v_1}(v_1) \ldots S_i^{v_m}(v_m))) \not\vdash \Omega)$

$\wedge\ \mathrm{svtraf}((p_i, p_{i+1}), (\Phi_i, S_i)) \Rightarrow (\Phi_{i+1}, S_{i+1}))$ und

iii) $\Phi_q \Rightarrow \psi(k)(S_q^{v_1}(v_1) \ldots S_q^{v_m}(v_m))$,

so ist kz partiell korrekt.

Beweis

Mit Hilfssatz

$(\forall\ 0 \leq i \leq q)((\Phi_i, S_i)$ ist bezüglich kz vorwärtsgültig an der Stelle i)

weitgehend analog zu Satz 10.

Bemerkung

U. a. wurde diese Methode in Deutsch [1973] und Burstall [1974] verwendet und von King [1975] im Rahmen einer Methodik des Programm-Testens diskutiert.
Analog zur symbolischen Vorwärts-Entwicklungs-Methode läßt sich noch eine "symbolische Rückwärts-Entwicklungs-Methode" darstellen, die jedoch für die Praxis der Programmverifikation bedeutungslos ist (siehe King [1976])

3.1.2.2 Terminierung

$\mathcal{P} = \{M_1, M_2, \ldots, M_{n(\mathcal{P})}\}$ sei weiterhin ein beliebiges Programm-System über $\mathfrak{A}$, $M_g \in \mathcal{P}$ beliebig und $M = M_g$ oder M ist das Hauptprogramm oder ein lokales Unterprogramm von M_g oder M ist ein beliebiger funktions-spezifizierter Modul von M_g oder ein paralleler Zweig eines parallelen Moduls von M_g oder ein Schleifen-Modul von M_g[1] oder der Schleifenkörper eines Schleifenmoduls von M_g.

$E_1 E_2 \ldots E_{n(M)}$ sei die operationelle Standardzerlegung von M und $G = (P, R, \alpha)$ der Standard-Strukturgraph von M.
J sei die vollständige Interpretation des Berechnungsalphabets und J' die vollständige Interpretation des Verifikationsalphabets von M_g.

---

1) Da Schleifen-Moduln keine Endzusicherung besitzen, konnte die Frage nach der partiellen Korrektheit von Schleifen-Moduln nicht gestellt werden. Die Frage nach der Terminierung von Moduln kann jedoch auch bei Schleifen-Moduln gestellt werden.

Def. 1

M terminiert $\Leftrightarrow$

$(\forall\ i \in ZS \setminus End)(\forall\ \sigma \in \Sigma_J)(\psi(i)^{J',\sigma} = W \rightarrow$

M terminiert bezüglich (i,σ)).

Satz 2

M sei kein Schleifen-Modul.
Ist M partiell korrekt und terminiert jeder Schleifen-Modul von M, so terminiert M.

Beweis

Seien $i \in ZS \setminus End$ und $\sigma \in \Sigma_J$ beliebig mit $\psi(i)^{J',\sigma} = W$.

Zu zeigen: $\exists$ B (B ist Berechnung auf M $\wedge$ anf(B) = (i,σ) $\wedge$

$P_1^2(end(B)) \in End$).

Sei B eine beliebige Berechnung mit anf(B) = (i,σ).

Falls $P_1^2(end(B)) \notin End$, so folgt aus der partiellen Korrektheit von M, daß B einen Folgezustand besitzt und da jeder Schleifen-Modul von M terminiert, läßt sich B zu einer Berechnung B' "verlängern" mit $P_1^2(end(B')) \in End$.

Bemerkung

Ähnlich wie bei der partiellen Korrektheit läßt sich auch die globale Eigenschaft der Terminierung eines Moduls M auf lokale Argumentationen zurückführen, d. h. hier auf den Nachweis der Terminierung der in M vorkommenden Schleifen-Moduln.
Die folgenden Überlegungen können sich somit auf die Frage nach der Terminierung eines beliebigen Schleifen-Moduls M_S von M beschränken.
M_{SK} sei hierbei der Schleifenkörper von M_S und in Abhängigkeit vom jeweiligen Kontext sei $E_1 \ldots E_{n(M)}$ die operationelle Standardzerlegung von $M = M_S$ bzw. $M = M_{SK}$, $G(M_S) = (P(M_S), R(M_S), \alpha(M_S))$ sei der Standardstrukturgraph von M_S und $G(M_{SK}) = (P(M_{SK}), R(M_{SK}), \alpha(M_{SK}))$ der Standardstrukturgraph von M_{SK}.

Lemma 3

a) $n(M_{SK}) = n(M_S)+1$.

b) $(\forall\ p \in M_{SK}\setminus\{1, n(M_{SK})-2, n(M_{SK})\})\ \alpha(M_{SK})(p) = \alpha(M_S)(p) \wedge$
$\alpha(M_{SK})(1) \neq \alpha(M_S)(1) \wedge \alpha(M_{SK})(n(M_{SK})-2) \neq \alpha(M_S)(n(M_S)-1)$.

Beweis

Siehe Definition von M_S und M_{SK}.

Def. 4

a) $\Sigma_{repeat} \leftrightharpoons \{\sigma \in \Sigma_J \mid \Phi_A(M_{SK})^{J',\sigma} = W \wedge \exists\ B\ (B$ ist Berechnung
auf $M_{SK} \wedge anf(B) = (1,\sigma) \wedge$
$P_1^2(end(B)) \in End\setminus\{n(M_{SK})\})\}$.

b) $\Sigma_{end} \leftrightharpoons \{\sigma \in \Sigma_J \mid \Phi_A(M_{SK})^{J',\sigma} = W \wedge \exists\ B\ (B$ ist Berechnung auf $M_{SK} \wedge$
$anf(B) = (1,\sigma) \wedge P_1^2(end(B)) = n(M_{SK}))\}$.

Lemma 5

$(\forall\ \sigma \in \Sigma_J)\ ((\sigma \in \Sigma_{end} \rightarrow \Phi_A(M_S)^{J',\sigma} = W \wedge$
$\exists\ B\ (B$ ist Berechnung auf $M_S \wedge anf(B) = (1,\sigma)$
$\wedge\ P_1^2(end(B)) = n(M_S))) \wedge$
$(\sigma \in \Sigma_{repeat} \rightarrow \Phi_A(M_S)^{J',\sigma} = W \wedge$
$\exists\ B\ (B$ ist Berechnung auf $M_S \wedge anf(B) = (1,\sigma)$
$\wedge\ P_1^2(end(B)) = 1)))$.

Beweis

Sei $\sigma \in \Sigma_J$ beliebig.

i) Sei $\sigma \in \Sigma_{end}$.
Da $\Phi_A(M_{SK})^{J',\sigma} = W$, gilt mit $\Phi_A(M_{SK}) = \Phi_A(M_S)$ auch $\Phi_A(M_S)^{J',\sigma} = W$.
Nach Voraussetzung gibt es ein $B = (z_o, z_1, \ldots, z_m)$ mit der Eigenschaft, daß B Berechnung auf M_{SK} und $z_o = (1,\sigma)$ und $P_1^2(end(B)) = n(M_{SK})$. $B' = (z_o, z_1, \ldots, z_{m-1})$ ist Berechnung auf M_S mit $P_1^2(end(B')) = n(M_S)$.

ii) Sei $\sigma \in \Sigma_{repeat}$.
Beweis analog zu i).

Bemerkung

Eine Berechnung auf M_S, die mit einer Belegung aus Σ_{repeat} beginnt, führt somit nach dem ersten Schleifendurchlauf auf den Eingang von M_S zurück und somit zu einer erneuten Berechnung auf dem Schleifenkörper von M_S, während eine Berechnung auf M_S, die mit einer Belegung aus Σ_{end} beginnt, unmittelbar (d. h. mit genau einer Berechnung auf dem Schleifenkörper von M_S) zum Ende von M_S führt.

Σ_{repeat} und Σ_{end} lassen sich mithilfe von G mechanisch bestimmen.

Lemma 6

a) $\Sigma_{repeat} \cap \Sigma_{end} = \emptyset$.

b) Ist M_{SK} partiell korrekt, so gilt:

$\Sigma_{repeat} \cup \Sigma_{end} = \{\sigma \in \Sigma_J \mid \Phi_A(M_{SK})^{J',\sigma} = W\}$.

Beweis

a) Für zwei Berechnungen B, B' auf M_{SK} mit anf(B) = anf(B') und $P_1^2(end(B)), P_1^2(end(B')) \in End$ gilt:

B = B', da modfu Funktion ist und in pf(B), pf(B') alle vorkommenden Punkte verschieden sind (es sind keine Schleifen möglich).

b) "$\subseteq$" gilt per Definition von Σ_{repeat} und Σ_{end}.

"$\supseteq$":

Sei $\sigma \in \Sigma_J$ beliebig mit $\Phi_A(M_{SK})^{J',\sigma} = W$.

Entsprechend dem Aufbau von M_{SK} folgt aus der partiellen Korrektheit:

$\exists$ B (B ist Berechnung auf $M_{SK} \wedge$ anf(B) = $(1,\sigma) \wedge$

$P_1^2(end(B)) \in End$),

d. h. $\sigma \in \Sigma_{repeat} \cup \Sigma_{end}$.

Satz 7

M_S terminiert, wenn gilt:

i) M_{SK} ist partiell korrekt und

ii) $(\forall \sigma \in \Sigma_{repeat})(\exists n \in N)(f(M_{SK}))^n(\sigma) \in \Sigma_{end}$ [1].

---

1) $f(M_{SK})$ ist die Bedeutung des Moduls M_{SK} (siehe Def. 8, Kap. 3.1.1.4) und $(f(M_{SK}))^n$ die n-fache Hintereinanderausführung von $f(M_{SK})$.

<u>Beweis</u>

Zu zeigen:

$(\forall\ \sigma \in \Sigma_J)(\Phi_A(M_S)^{J',\sigma} = W \rightarrow$

$\exists\ B$ (B ist Berechnung auf $M_S \wedge \mathrm{anf}(B) = (1,\sigma) \wedge$

$P_1^2(\mathrm{end}(B)) = n(M_S)))$.

Sei $\sigma \in \Sigma_J$ beliebig mit $\Phi_A(M_S)^{J',\sigma} = W$.

Da $\Phi_A(M_S) = \Phi_A(M_{SK})$ gilt mit der partiellen Korrektheit von M_{SK} nach Lemma 6b), daß $\sigma \in \Sigma_{repeat} \cup \Sigma_{end}$.

Falls $\sigma \in \Sigma_{end}$, so gilt mit Lemma 5 die Behauptung.

Sei also $\sigma \in \Sigma_{repeat}$.

Nach Voraussetzung existiert ein $n \in N$ mit

$(f(M_{SK}))^n(\sigma) \in \Sigma_{end}$, d. h. es gilt:

M_{SK} terminiert bezüglich $(1,(f(M_{SK}))^n(\sigma)$ und für alle $1 \leq i \leq n$ gilt:

M_{SK} terminiert bezüglich $(1,(f(M_{SK}))^i)$, wobei
$\mathrm{val}(M_{SK},(1,(f(M_{SK}))^i(\sigma))) = f(M_{SK})((f(M_{SK}))^i(\sigma))$.

Somit gilt mit Lemma 5:

$\exists\ B_1$ (B_1 ist Berechnung auf $M_S \wedge \mathrm{anf}(B_1) = (1,\sigma) \wedge$

$P_1^2(\mathrm{end}(B_1)) = 1) \wedge$

und es existieren Berechnungen $B_2,\ldots,B_{n+1}$ auf M_S, wobei

für $2 \leq i \leq n$ gilt:

$\mathrm{anf}(B_i) = (1,P_2^2(\mathrm{end}(B_{i-1}))) \wedge P_1^2(\mathrm{end}(B_i)) = 1)$ und

$\mathrm{anf}(B_n) = \mathrm{end}(B_{n-1})$.

Mit $P_2^2(\mathrm{end}(B_n)) \in \Sigma_{end}$ gilt: $P_1^2(\mathrm{end}(B_n)) = n(M_S)$.

Wir erhalten also eine Berechnung B auf M_S mit

$\mathrm{anf}(B) = (1,\sigma)$ und $P_1^2(\mathrm{end}(B)) = n(M_S)$.

Satz 8

Ist M_{SK} partiell korrekt, so gilt mit

$$f_{Schleife} \doteq \left\langle \sigma \mapsto \begin{cases} f(M_{SK})(\sigma), \text{ falls } \sigma \in \Sigma_{repeat} \\ \Omega \text{ sonst} \end{cases} \middle| \sigma \in \Sigma_J \cup \{\Omega\} \right\rangle$$

$$(\forall\ \sigma \in \Sigma_{repeat})(\forall\ n \in N)(f^n_{Schleife}(\sigma) = \Omega \rightarrow (\exists\ m \in N)(1 \leq m < n \wedge f^m(M_{SK})(\sigma) \in \Sigma_{end})).$$

Beweis

Seien $\sigma \in \Sigma_{repeat}$ und $n \in N$ beliebig mit $f^n_{Schleife}(\sigma) = \Omega$.

Annahme:

$\neg(\exists\ m \in N)(1 \leq m < n \wedge f^m(M_{SK})(\sigma) \in \Sigma_{end})$, d. h.

$(\forall\ m \in N)(m \geq n \vee f^m(M_{SK})(\sigma) \notin \Sigma_{end})$.

Für beliebige m mit $m<n$ gilt somit: $f^m(M_{SK})(\sigma) \notin \Sigma_{end}$.

Beh.: $f^m(M_{SK})(\sigma) \in \Sigma_{repeat}$ für alle $m<n$.

Bew.:

i) $m = 1$: Da M_{SK} partiell korrekt und $\sigma \in \Sigma_{repeat}$ gilt:

$f_{Schleife}(\sigma) = f(M_{SK})(\sigma)$ und

$\Phi_A(M_{SK})^{J', f_{Schleife}(\sigma)} = W$.

Mit Lemma 6b) und der Voraussetzung $f(M_{SK})(\sigma) \notin \Sigma_{end}$ gilt somit: $f(M_{SK})(\sigma) \in \Sigma_{repeat}$.

ii) Sei $f^{m'}(M_{SK})(\sigma) \in \Sigma_{repeat}$ für $1 \leq m' < m$.

Beh.: $f^{m'+1}(M_{SK})(\sigma) \in \Sigma_{repeat}$.

Bew.: Analog zu i) gilt: $f_{Schleife}(f^{m'}(M_{SK})(\sigma)) =$

$f(M_{SK})(f^{m'}(M_{SK})(\sigma)) = f^{m'+1}(M_{SK})(\sigma)$ und

$\Phi_A(M_{SK})^{J', f^{m'+1}(M_{SK})(\sigma)} = W$.

Somit gilt wiederum nach Lemma 6b) mit der Voraussetzung $f^{m'+1}(M_{SK})(\sigma) \notin \Sigma_{end}$, daß $f^{m'+1}(M_{SK})(\sigma) \in \Sigma_{repeat}$.

Gilt $f^{n-1}(M_{SK})(\sigma) \in \Sigma_{repeat}$, so gilt wiederum

$\Phi_A(M_{SK})^{J',f^n(M_{SK})(\sigma)} = W$ und damit

$f^n(M_{SK})(\sigma) \in \Sigma_{repeat} \cup \Sigma_{end}$, d. h. $f^n(M_{SK})(\sigma) \neq \Omega$

im Widerspruch zur Voraussetzung.

Somit gilt die Negation der Annahme, d. h.

$(\exists\ m \in N)(1 \leq m < n \wedge f^m(M_{SK})(\sigma) \in \Sigma_{end})$.

<u>Bemerkung</u>

Satz 7 und Satz 8 liefern die Grundlage für eine Methode zum Nachweis der Terminierung von Schleifen-Moduln M_S:

i) Beweise die partielle Korrektheit des Schleifenkörpers M_{SK} von M_S, d. h. ermittle eine "passende" Schleifeninvariante $\Phi_A(M_S)$, auf die sich der Nachweis der Terminierung beziehen kann.

ii) Bestimme die Mengen Σ_{repeat} und Σ_{end} sowie $f(M_{SK})|\Sigma_{end}$, z. B. mithilfe der Vorwärts-Entwicklung von symbolischen Ausdrücken entlang der Kantenzüge, die von 1 nach $n(M_{SK})$ bzw. von 1 nach $End \setminus \{n(M_{SK})\}$ führen.

iii) Ermittle für ein beliebiges $\sigma \in \Sigma_{repeat}$ ein $n(\sigma) \in N$ mit $f^{n(\sigma)}_{Schleife}(\sigma) \notin \Sigma_{repeat}$, d. h. eine <u>Terminierungsfunktion</u>.

iv) Nach Ausführung der Schritte i) - iii) gilt mit Satz 8 und Satz 7, daß M_S terminiert.

Die Mengen Σ_{repeat} und Σ_{end} drückt man dabei zweckmäßigerweise direkt mittels $\Phi_A(M_S)$ aus, indem man $\Phi_A(M_S)$ als Formel $(\Phi_{repeat} \vee \Phi_{end})$ darstellt mit

$\{\sigma \in \Sigma_J | \Phi^{J',\sigma}_{repeat} = W\} = \Sigma_{repeat}$ und $\{\sigma \in \Sigma | \Phi^{J',\sigma}_{end} = W\} = \Sigma_{end}$.

3.2 Architektonische Modellbildung

Gegeben sei eine beliebige Menge $\tilde{\mathcal{F}}$ berechenbarer Funktionen und eine Menge $\mathfrak{A}$ von Anwenderbereichen.
Für alle $(A,J) \in \mathfrak{A}$ gelte:
Zu jedem $f \in FKonst(A)$ gibt es ein Rechensystem $RS((A,J))$, das für beliebige Argumente $\bar{x} \in Dom(J(f))$ den Wert $J(f)(\bar{x})$ berechnet.
Gesucht ist ein Rechensystem $RS(\tilde{\mathcal{F}})$, das für beliebige Eingaben $(f,\bar{x})$ mit $f \in \tilde{\mathcal{F}}$ und $\bar{x} \in Dom(f)$ den Wert $f(\bar{x})$ berechnet.

Mit der in den vorausgegangenen Abschnitten entwickelten algorithmischen Modellbildung läßt sich eine derartige Aufgabenstellung zunächst auf die Entwicklung eines korrekten Programm-Systems $\mathcal{P} = \{M_1, M_2, \ldots, M_n\}$ über $\mathfrak{A}$ reduzieren, wobei gelten muß:
$(\forall f' \in \tilde{\mathcal{F}})(\exists\ 1 \leq i \leq n)\ f(H(M_i), \Phi_A(M_i)) = f'$ (wobei $f(H(M_i), \Phi_A(M_i))$ die Bedeutung des Hauptprogramms $H(M_i)$ von M_i bezüglich der Anfangszusicherung $\Phi_A(M_i)$ von M_i ist).

Die Frage nach einer Realisierung der der algorithmischen Modellbildung zugrundeliegenden operationellen Semantik wurde dabei bisher ausgeklammert und soll nun unter den oben skizzierten Randbedingungen aufgegriffen werden.
Dabei wollen wir uns nicht auf konkrete technologische Bausteine beziehen, sondern mithilfe abstrakter Prozessor-, Bus- und Speicherbausteine (deren Struktur und Arbeitsweise offen und damit an die jeweils zweckmäßigste Technologie anpaßbar bleibt) ein architektonisches Modell skizzieren, auf dem eine technologische Realisierung (bzw. detailliertere architektonische Modellierung) der Bedeutung unserer Programmsysteme heute vorteilhaft aufbauen könnte.

Die historisch gesehen naheliegendste Realisierungsmöglichkeit für die Bedeutung unserer Programmsysteme liegt in der Verwendung eines von-Neumann-Computers, dessen Fähigkeiten mittels Compiler und anderer Systemprogramme an die semantische Ebene unserer Programmsysteme anzugleichen wären.
Das architektonische Modell eines von-Neumann-Computers besteht grob betrachtet aus einem Speicher, einem Prozessor und einem Bus, wobei im Speicher neben den jeweils aktuellen Berechnungszuständen auch die (in eine interne Darstellung übersetzten) auszuführenden Programme stehen, der Prozessor jeweils genau einen Berechnungszustand genau eines Programm-Moduls bearbeitet, d. h. die nächste auszuführende Moduloperation ermittelt und bei Assignationen und globalen Prozeduraufrufen den Speicherinhalt verändert, und der Bus schrittweise Speicherwörter vom Speicher zum Prozessor bzw. vom Prozessor zum Speicher überträgt.

Bildliche Darstellung:

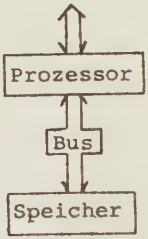


Abb. 1: Modell eines von-Neumann-Computers.

Eine derartige Realisierung der Bedeutung von Programmen führt zu der an vielen Stellen beschriebenen und kritisierten "Sequentialisierung und Detaillisierung"[1] der Ausführung von Aufträgen durch das Rechensystem und verhindert nicht nur die parallele Bearbeitung mehrerer anstehender Aufträge, sondern

---

1) Siehe z. B. Bauer [1978] und Backus [1978].

bedingt auch eine sequentielle Ausführung der parallelen Zweige paralleler Moduln und eine Zerlegung der Modul-Berechnungsfunktion modfu in sequentielle Berechnungsschritte. Betrachten wir z. B. die Berechnung von modfu((i,σ)), wenn α(i) eine Assignation der Form

$((t_1 \Rightarrow (v_{11},\ldots,v_{1n_1})),\ldots,(t_m \Rightarrow (v_{m1},\ldots,v_{mn_m})))$.

Eine parallele, gleichzeitige Berechnung der Werte von $t_1,t_2,\ldots,t_n$ und Zuweisung der berechneten Werte an die Variablen $v_{11},v_{12},\ldots,v_{mn_m}$ ist bei obigem Realisierungs-Modell nicht möglich und selbst die Berechnung des Wertes eines Terms t_i zerfällt hierbei in der Regel in viele sequentielle Einzelschritte, wobei zusätzliche Speicherplätze zur Abspeicherung von Zwischenergebnissen benötigt werden.
Obwohl in der Terminologie von Backus [1978] unsere Programme mit der Verwendung von Variablen und Wertzuweisungen in einer von-Neumann-Sprache formuliert sind, ist das obige Modell des von-Neumann-Computers für eine Realisierung der Semantik unserer Programme ähnlich ungeeignet, wie das Modell der Turing-Maschine zur Realisierung der Semantik von z. B. ALGOL 60-Programmen.
Mit der Entwicklung geeigneter Modelle (bzw. Rechnerarchitekturen) für die Realisierung der operationellen Semantik unserer "von-Neumann-ähnlichen" Sprachen würde der Kritik von Backus an von-Neumann-Sprachen partiell die Grundlage entzogen.

Die monolithische Struktur des von-Neumann-Computers widerspricht auch den im Kapitel 3.1.1.1 dargestellten Grundprinzipien zur Strukturierung ingenieurwissenschaftlicher Modellbildungen. Es gibt keinen vernünftigen Grund dafür, die Forderung nach modularem, klar und übersichtlich strukturiertem, leicht erweiterbarem Systemaufbau auf die algorithmische Modellbildung zu beschränken, zumal uns die technologische Entwicklung billige und kleine LSI-Bausteine zur Realisierung modular aufgebauter architektonischer Modelle zur Verfügung stellt.

An Post-von-Neumann-Computern wird in Erlangen schon seit längerem gearbeitet und unsere Suche nach einer geeigneten architektonischen Modellbildung kann auf Überlegungen zum EGPA-Projekt (siehe z. B. Händler/Hofmann/Schneider [1974]) aufbauen: Auch unser Modell eines Post-von-Neumann-Computers wird auf einer Vielzahl hierarchisch strukturierter Prozessoren aufbauen, zwischen lokalen und globalen Speichern unterscheiden, die kommerziell verfügbare Hardware berücksichtigen und eine dynamische Anpassung der Modellstruktur an unterschiedliche Anforderungen ermöglichen.

Der Semantikdefinition unserer Programmsysteme liegt der Begriff der Modulberechnungsfunktion modfu = (indexfu,belegfu) zugrunde, wobei indexfu den "Index" der als nächstes auszuführenden elementaren Moduloperation und belegfu ggf. eine neue Belegung der Modulvariablen liefert. Die Anforderungen an eine Realisierung von indexfu und belegfu sind somit völlig verschieden: indexfu ist bei gegebener Auswertung von Verzweigungsbedingungen unabhängig vom jeweiligen Anwenderbereich, während die Realisierung von belegfu (und die Auswertung von Verzweigungsbedingungen) vom jeweiligen Anwenderbereich abhängig ist. Es ist somit naheliegend die Realisierung von indexfu und belegfu zwei verschiedenen Prozessoren, einem Indexfu- und einem Belegfu-Prozessor, zuzuordnen.
Nicht nur diese Aufspaltung der auszuführenden Arbeit auf zwei Prozessoren, sondern auch die unterschiedlichen Anforderungen an die Struktur des Befehls- und Wertespeichers[1)] legen außerdem eine Aufspaltung der zu speichernden Information in einen Befehls- und einen Wertespeicher nahe.
Neben internen Verbindungen zwischen Indexfu- und Belegfu-Prozessor (Bus 1), zwischen Indexfu-Prozessor und Befehlsspeicher (Bus 2) sowie zwischen Belegfu-Prozessor und Wertespeicher (Bus 3) werden noch externe Verbindungen benötigt:

---

1) Die Struktur des Befehlsspeichers kann wieder als anwendungsinvariant angesehen werden, während es vorteilhaft ist, die Struktur des Wertespeichers dem jeweiligen Anwenderbereich (d. h. den den Bereichskonstanten zugeordneten Objekten) anzupassen.

Eine Verbindung zum Übermitteln von Aufträgen bzw. Fehlermeldungen (S1), eine Verbindung zum Ein- und Auslesen von Prozeduren (S2) und eine Verbindung zum Ein- und Auslesen von Werten (S3).
Damit haben wir die Berechnung von Programm-Moduln auf das folgende architektonische Modell zurückgeführt:

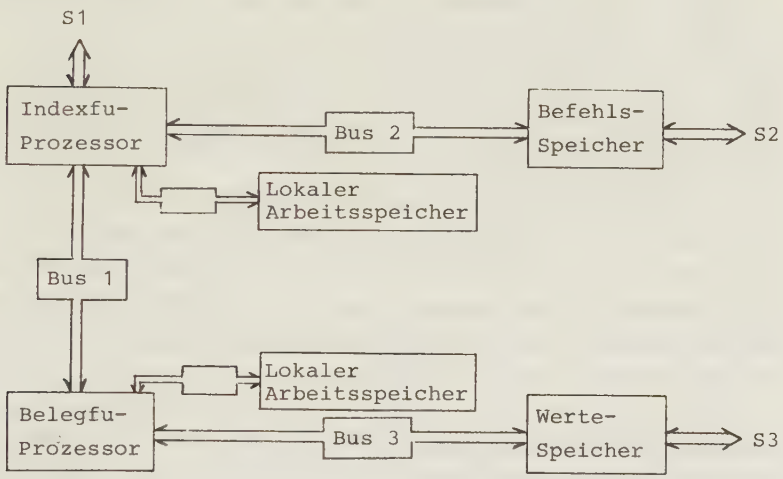


Abb. 2: Modell eines Modul-Berechnungs-Elements.

Mit der Anpassung des Belegfu-Prozessors, der Bedeutung der Befehle im Befehlsspeicher und des Wertespeichers an einen konkreten Anwenderbereich (A,J) können wir von dem einem Modul-Berechnungs-Element E zugrundeliegenden Anwenderbereich sprechen, wobei Dom(E) den E zugrundeliegenden Anwenderbereich bezeichne.

Bei der nun folgenden groben Skizzierung der Arbeitsweise eines Modul-Berechnungs-Elements (kurz MBE) wollen wir uns auf den Fall beschränken, daß das MBE immer nur höchstens an einem Auftrag arbeitet[1].
Bevor über S1 dem MBE ein Auftrag mitgeteilt wird, sei bereits über die Schnittstelle S2 die auszuführende Prozedur - unter Etablierung einer festen Beziehung zwischen den Prozedurvariablen einerseits und den Adressen des Wertespeichers andererseits - in den Befehlsspeicher eingelesen worden; außerdem seien bereits über die Schnittstelle S3 in die den Eingabevariablen zugeordneten Speicherplätze die Anfangswerte der Eingabevariablen eingelesen. (Die Verwaltung des Befehls- und Wertespeichers liege außerhalb des MBE's.)
Die auszuführende Prozedur sei dabei natürlich in einer für den Prozeß der Berechnung zweckmäßigen internen Darstellungsweise gegeben. Insbesondere seien lokale Prozeduraufrufe durch eine Inkarnation des jeweils zugeordneten Moduls ersetzt sowie all die Information entfernt worden, die nur zur Erleichterung der Programmverifikation dient.

Nach einer Auftragsmitteilung über S1 bzw. einer Fertigmeldung vom Belegfu-Prozessor liest der Indexfu-Prozessor über den Bus 1 einen Befehl aus dem Befehlsspeicher, entschlüsselt den Befehl und teilt ggf. über den Bus 2 dem Belegfu-Prozessor mit, wie (d. h. mit welchen Werten, Operationen und Berechnungsschritten) welche Speicherplätze im Wertespeicher zu verändern sind (bei Assignationen sowie bei globalen Prozeduraufrufen) bzw. welche wie zu ermittelnden Werte miteinander zu vergleichen sind (bei Verzweigungen).
In Abhängigkeit vom vorliegenden Befehl b und ggf. der auf diesen Befehl folgenden Antwort a des Belegfu-Prozessors sind die folgenden sieben Fälle zu unterscheiden:

---

1) Für die gleichzeitige Bearbeitung mehrerer Aufträge benötigt der Indexfu-Prozessor eine Warteschlange für von außen bzw. vom Belegfu-Prozessor eintreffende Aufträge bzw. Rückmeldungen und der Belegfu-Prozessor eine Warteschlange für vom Indexfu-Prozessor übermittelte Anweisungen. Die Handhabung dieser Warteschlangen und der damit verbundenen Koordinierungsarbeiten würde die Darstellung unserer architektonischen Modellbildung zusätzlich komplizieren.

i) b spezifiziert eine Assignation:
a ist eine Fertig- oder eine Fehlermeldung.
Bei einer Fertigmeldung liest der Indexfu-Prozessor den durch b spezifizierten Folgebefehl aus dem Befehlsspeicher. Eine Fehlermeldung wird vom Indexfu-Prozessor ggf. modifiziert und über S1 nach außen weitergeleitet. Anschließend wartet Indexfu (und damit das MBE) auf einen neuen Auftrag. Fehler bedingen somit keinen System-Ausfall.

ii) b spezifiziert eine dezentrale Verzweigung:
a ist eine Fertigmeldung mit der Spezifikation der beiden Alternativen "W" oder "F" oder eine Fehlermeldung.
Bei einer Fertigmeldung liest der Indexfu-Prozessor den durch b und a spezifizierten Folgebefehl aus dem Befehlsspeicher.
Eine Fehlermeldung wird wie unter i) bearbeitet.

iii) b spezifiziert eine zentrale Verzweigung:
a ist eine Fertigmeldung mit der Spezifikation der Alternative oder eine Fehlermeldung.
Bei einer Fertigmeldung liest der Indexfu-Prozessor den durch b und a spezifizierten Folgebefehl aus dem Befehlsspeicher.
Eine Fehlermeldung wird wie unter i) bearbeitet.

iv) b spezifiziert einen globalen Prozeduraufruf der Form
<u>gcall</u> name $((t_1,t_2,\ldots,t_m),(v_1,v_2,\ldots,v_n))$.
a ist eine Fertigmeldung oder eine Fehlermeldung.
Bei einer Fehlermeldung wird wie unter i) verfahren und bei einer Fertigmeldung stellt der Indexfu-Prozessor an der Schnittstelle S1 eine im Befehl b enthaltene Kennung bereit, die den aufzurufenden Modul name, zwei Adresslisten des Wertespeichers (aktuelle Parameter) und eine Adresse des Befehlsspeichers (Rücksprungadresse) spezifiziert. Da Werte- und Befehlsspeicher außerhalb des MBE's verwaltet werden, wird davon ausgegangen, daß diese Adresslisten und die Rücksprungadresse unter der im Befehl b enthaltenen Kennung außerhalb des Rechensystem-Elements bereits zur Verfügung stehen.

Nachdem der an der Schnittstelle S1 bereitgestellte Prozeduraufruf abgeholt ist (und die aktuellen Parameter über S3 aus dem Wertespeicher gelesen werden) wartet das Rechensystem-Element auf einen neuen Auftrag.
Ist der Prozeduraufruf ausgeführt, so werden über S3 die Ergebnisse unter der jeweils zugeordneten Adresse in den Wertespeicher geladen und anschließend erhält das MBE über S1 den Auftrag, beginnend mit der "Rücksprungadresse" die Befehle im Befehlsspeicher weiterzubearbeiten.

v) b spezifiziert die Beendigung eines vollständig spezifizierten Moduls:
Der Indexfu-Prozessor stellt an S1 eine mit einer Kennung des beendeten Moduls versehene "Fertigmeldung" bereit (die im Befehl b bereits enthalten sei), worauf die der Prozedur zugeordneten Ausgabeparameter aus dem Wertespeicher gelesen werden und ggf. der der Prozedur zugewiesene Platz im Befehls- und Wertespeicher freigegeben wird. Anschließend wartet das Rechensystem-Element auf einen neuen Auftrag.

vi) b spezifiziert den Anfang eines parallelen Moduls, der n parallele Zweige $z_1, z_2, \ldots, z_n$ besitze:
Es sei vorausgesetzt, daß jeder der den Zweigen $z_1, z_2, \ldots, z_n$ zugeordneten Prozeduren (einschließlich der jeweils vorkommenden lokalen Unterprogramme) ein MBE zugeordnet sei, d.h. jede dieser Prozeduren in dem Befehlsspeicher eines derartigen Systems steht. Außerdem sei in dem die Befehls- und Wertespeicher verwaltenden System jeweils eine "Umrechnungsvorschrift" zwischen den Adressen der Variablen von z_i im z_i ausführenden MBE und den Adressen dieser Variablen im gegebenen "übergeordneten" MBE etabliert worden.
Der Indexfu-Prozessor stellt an S1 eine im Befehl b enthaltene Kennung für die parallelen Zweige $z_1, z_2, \ldots, z_n$ zur Verfügung, worauf über S3 die Werte der Variablen von $z_1, z_2, \ldots, z_n$ aus dem Wertespeicher gelesen werden. Danach wartet das MBE auf einen neuen Auftrag (der in der Regel aus der Ausführung eines der parallelen Zweige z_i bestehen wird).

Sind alle parallelen Zweige bearbeitet, so werden über S3 die Ergebnisse dieser voneinander unabhängigen Teilberechnungen in den Wertespeicher gelesen und anschließend erhält das MBE über S1 den Auftrag, beginnend mit dem auf b folgenden Befehl, die Befehle im Befehlsspeicher weiter zu bearbeiten.

vii) b spezifiziert das Ende eines parallelen Zweiges:
Der Indexfu-Prozessor stellt - völlig analog zur Beendigung vollständig spezifizierter Moduln - an S1 eine mit der Kennung des beendeten Moduls versehene "Fertigmeldung" bereit, worauf die Werte der modifizierbaren Variablen des Moduls aus dem Wertespeicher gelesen und ggf. der dem Modul zugewiesene Platz im Befehls- und Wertespeicher freigegeben wird. Anschließend wartet das MBE auf einen neuen Auftrag.

Das angestrebte Rechensystem RS($\mathfrak{T}$) bestehe bisher aus dem Programmsystem $\mathcal{P}$ und einer gewissen Anzahl von Modul-Berechnungs-Elementen, wobei für jedes $M \in \mathcal{P}$ mindestens ein MBE e existiere mit Dom(e) = Dom(M)[1].
Die weitere Realisierung von RS($\mathfrak{T}$) wollen wir auf folgendes architektonisches Rechensystem-Modell beziehen:

---

1) Die Anzahl an Modul-Berechnungs-Elementen e mit Dom(e) = Dom(M) für ein beliebiges, festes $M \in \mathcal{P}$ orientiert sich hierbei an dem Umfang und der Art der zu verarbeitenden Aufträge, an der Struktur des Programmsystems (vorkommende parallele Moduln) sowie an zu erfüllenden Antwortzeiten (Durchsatzgeschwindigkeit) und der geforderten Ausfallsicherheit.

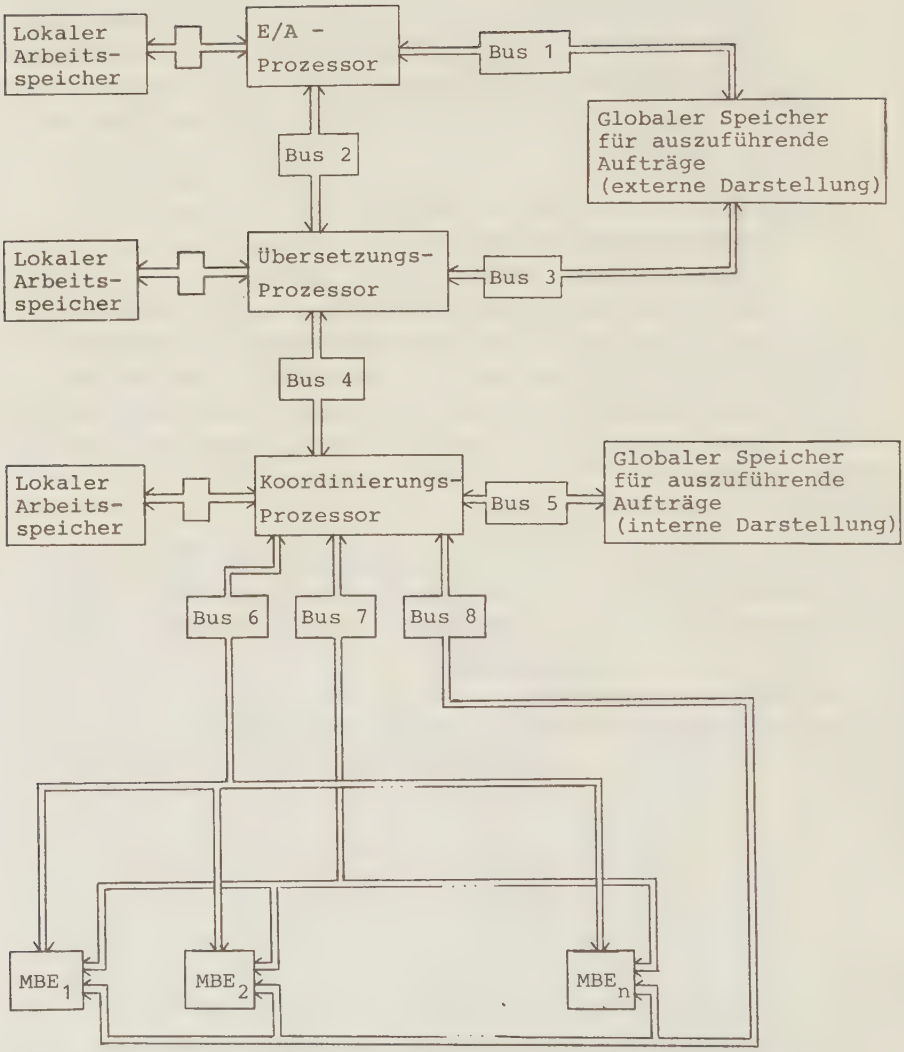


Abb. 3: Eine auf Anwenderbereichen aufbauende architektonische Rechensystem-Modellbildung.

Der E/A-Prozessor liest über den Bus 1 das Programm-System $\mathcal{P}$ in einen globalen Speicher und startet anschließend über den Bus 2 den Übersetzungs-Prozessor, der das Programm $\mathcal{P}$ in eine interne Darstellung mit einem vom Anwendungsbereich unabhängigen Befehlsformat übersetzt, wobei zwischen der internen Darstellung und dem Programmtext P Bezüge hergestellt und vermerkt werden[1]. Das übersetzte Programm wird über den Bus 4, den Koordinierungs-Prozessor und den Bus 5 in einem globalen Speicher abgelegt.
Erhält nun der E/A-Prozessor einen Auftrag $(M,\bar{x})$, wobei $M \in \mathcal{P}$ und $\bar{x} = \{(v,\text{wert}) \mid v$ ist Komponente der Eingabeparameterliste von M und wert $\in J(b(v,M))\}$,
so wird dieser Auftrag über den Bus 2 dem Übersetzungs-Prozessor mitgeteilt, der diesen Auftrag in eine interne Darstellung übersetzt und diese dem Koordinierungs-Prozessor mitteilt. Der Koordinierungs-Prozessor koordiniere bereits die Ausführung anderer Aufträge, wobei er über den jeweils aktuellen Systemzustand (Art und Anzahl der intakten MBE_i's sowie deren Speicherbelegung und Auslastungsquote, Art und Anzahl der gegenwärtig (und in naher Zukunft) zu bearbeitenden Aufträge und deren jeweilige Priorität) Buch führe. Unter Berücksichtigung des aktuellen Systemzustandes beginnt der Koordinierungs-Prozessor mit der Zerlegung des globalen Auftrages in aufeinanderfolgende Teilaufträge und der Übermittlung der ersten auszuführenden Prozedur an ein ausgewähltes MBE (Laden von Programm- und Werte-Speicher des MBE's über Bus 7 bzw. 8 und Aktivierung des MBE's unter Mitteilung der Startadresse über Bus 6).
Die weitere Ausführung des Auftrags $(M,\bar{x})$ geschieht nun in einem Wechselspiel zwischen Teilaufträge bearbeitenden MBE's und dem Koordinierungsprozessor, wobei die Auswahl der zu beauftragenden MBE's jeweils von aktuellen Systemzustand abhängig sei.

---

1) Dieser Bezug kann neben der Generierung lesbarer Fehlermeldungen auch einer interaktiven Programm-Modifikation bzw. Programmerstellung (einschließlich Programmverifikation) dienen. Die Frage nach Einsatzmöglichkeiten unseres architektonischen Rechensystem-Modells für den Prozess der interaktiven Programmierung bzw. hierfür notwendigen Erweiterungen/Modifikationen der Modell-Struktur wollen wir hier jedoch ausklammern.

Ist der Auftrag $(M,\bar{x})$ bearbeitet, so übermittelt der Koordinierungs-Prozessor das Ergebnis $\bar{y}$ (bzw. eine Fehlermeldung) über eine Rückübersetzung an den E/A-Prozessor, der das Ergebnis (bzw. die Fehlermeldung) ausgibt.

Zusammengefaßt lassen sich einige wesentliche Eigenschaften unserer auf Anwenderbereichen aufbauenden architektonischen Rechensystem-Modellbildung folgendermaßen skizzieren:

1) Bei der Verwendung hinreichend vieler, passend gewählter Modul-Berechnungs-Elemente besteht kein Zwang zur Sequentialisierung und Detaillisierung bei der Ausführung von Aufträgen. Die in unseren Rechenprogrammen enthaltenen Möglichkeiten zur Parallelverarbeitung können ggf. voll ausgenutzt werden.

2) Durch seinen modularen Aufbau ist unser Modell relativ leicht sich ändernden Randbedingungen (z. B. Durchsatz) anpaßbar. Auftretende Engpässe lassen sich "sanft", d. h. ohne Übergang zu einem völlig neuen System, meistern, indem z. B. zusätzliche MBE's angeschlossen werden, oder das System unter Verwendung mehrerer Koordinierungs-Prozessoren in mehrere hierarchisch gegliederte Teilsysteme zerlegt wird.

3) Auftretende Software- und Hardware-Fehler lassen sich relativ leicht lokalisieren und bedingen auf der Ebene der MBE keinen System-Zusammenbruch. Durch die Vervielfachung ausgewählter Bausteine läßt sich die Ausfallsicherheit gezielt erhöhen.

4) Für die Realisierung der Funktion von Belegfu-Prozessoren können ggf. besonders schnell arbeitende spezielle Hardware-Bausteine entwickelt werden (z. B. "Datenfluß-Rechner").

4. Stichwortverzeichnis

abblockend 92, 112

Anweisung
- goto ~ 38,39
- if-then-else ~ 54,61
- selektive ~ 60

Anwenderalphabet 16

Anwenderbereich 19

anwenderorientiert 4

anwendungsinvariant 8

anwendungsspezifisch 8

Assignation 44

Belegung 21

Berechnung 2,97

Berechnungsalphabet 80

Berechnungsbereich 80

Berechnungssymbole 40

Berechnungszustand 92,97

Bereichsnamen 41

Blockstruktur 34

Bus-Baustein 145

Datenflußrechner 156

Datentyp 11

EGPA-Projekt 148

Einfachzweigabschluß 85

Einfachzweigsammel-
punkt 85

Endpunkt 87

Entwurf 4

Euclid 7

Formel 25
- rückwärtsgültige ~ 101,105
- rückwärtsgültige ~
bzgl. Kantenzug 125
- Rückwärtstransformation
einer ~ 118
- vorwärtsgültige ~ 101,105
- vorwärtsgültige ~
bzgl. Kantenzug 124
- Vorwärtstransformation
einer ~ 110

Funktionsprozedur 35

Gleichung 26

Hauptprogramm 74,80,83

Hierarchisierung 32

Implementierung 7

Interpretation 15,18,19

Kantenzug
- einfacher, gerichteter ~ 90
- partiell korrekter ~ 124

Kommentar-Alphabet 41

Konstante
- Bereichs ~ 16
- Funktions ~ 16
- Individuen ~ 16
- Relations ~ 14

Kontrollzeichen 40

Logik 5,8,14

LSI-Baustein 1,147

Marken 57,84

Modul 31
- dezentral-verzweigender ~ 53
- einfach-serieller ~ 38,49
- funktions-spezifizierter ~ 35,69
- global-aufrufbarer ~ 79
- kommentierter ~ 35,69
- lokal-aufrufbarer ~ 71
- nackter ~ 35,68
- paralleler ~ 50,62
- partiell-korrekter ~ 104
- partiell-spezifizierter ~ 35,73
- vollständig-spezifizierter ~ 35,77
- zentral-verzweigender ~ 55

Modulbedeutung 98

Modulberechnungs-Element 149

Modulberechnungsfunktion 95,148

Modulkopf 69,70,72,79,84
- vollständiger ~ 75

Modulname 41

Moduloperation
- elementare ~ 86

Modulschachtelung 32,34
Modulspezifikation 35
Modul-Terminierung 138
Modul-Verknüpfungs-
struktur 39
Multiprozessor 1,148

operationelle Standard-
zerlegung 87

paralleler Zweig 51,147,
152,153
Parallelverarbeitung 8,152,
156
Parallelzweigabschluß 85
Parallelzweigsammelpunkt 85
partielle Korrektheit 100,104
Problemlösen 6
Programm 83
Programmlokalität 32,39,58
Programmschema 9
Programm-System 82
Programmierlaboratorium 8
Prozedur
- globale ~ 33,77
- lokale ~ 33,71
Prozeduraufruf
- globaler ~ 47
- lokaler ~ 46
Prozedurende 85
Prozessor 145,148,155

Rechenaufgabe 2
Rechensystem 2,3,4,11
- von-Neumann-ähnliches ~ 3
Rechnerarchitektur 4,5,146,147,
148,154,156

Schleifen-Eingang 85
Schleifen-Körper 65
Schleifen-Modul 63
Schleifen-Rücksprung 85
Semantik
- denotationelle ~ 10,98
- operationelle ~ 5,92-98
symbolischer Ausdruck 129

Term 22
Terminierung 97,102,138
Terminierungsfunktion 144

Unterprogramm
- lokales ~ 74,79,83
- globales ~ 83
Unterprogramm-System 73

Variable
- Individuen ~ 16
- Modul ~ 45,77
Variableninterpretation 75
Verifikation 6,100
Verifikationsalphabet 80
Verifikationsbedingung 129
Verifikationsbereich 79
Verifikationskalkül
- ~ zur Entwicklung vorwärtsgültiger Formeln 112
- ~ zur Entwicklung rückwärtsgültiger Formeln 120
- ~ zur Entwicklung von bzgl. Kantenzügen vorwärtsgültigen Formeln 125
- ~ zur Entwicklung von bzgl. Kantenzügen rückwärtsgültigen Formeln 127
- ~ zur Entwicklung von bzgl. Kantenzügen vorwärtsgültigen symbolischen Ausdrücken 132
Verifikationsmethode 101,102,
108-144
von-Neumann-Computer 146
von-Neumann-Sprache 3,147

weakest pre-condition 101,129
Wertzuweisung 43
- deterministische ~ 38

Zielverteiler 60
Zusicherung 7,51,64,70,79,85

5. Literaturverzeichnis

Aho, A./ Ullmann, J.: "The theory of parsing, translation and compiling, Volume 1: Parsing", Prentice-Hall, Inc. Englewood Cliffs, N. Y. 1972

Altmann, W.: "Beschreibung von Programm-Moduln zum Entwurf zuverlässiger Software", Arbeitsbericht des IMMD, Bd. 11, Nr. 16, Erlangen 1978

Balzert, H.: "Systementwicklung, hierarchische Strukturen, Modularisierung, Datenabstraktion", Bericht 21 des FB Informatik der Universität Kaiserslautern, Dezember 1977

Basu, S. K./ Yeh, R. T.: "Strong verification of programs", Soft. Eng. and Syst. Lab., Univ. of Texas - Austin, Techn. Report SESLTR - 13, 1975

Bauer, F. L.: "Design of a programming language for a program transformation system", Working Material of the "International Summer School on Program Construction", Munich, July 26 to August 6, 1978

Bauer, F. L.: "Vorlesungsskriptum ALGORITHMISCHE SPRACHEN", Technische Universität München, Institut für Informatik, Juli 1977

Berkling, K.: "Computer architecture for correct programming", "The 5th Annual Aymposium on Computer Architecture", pp. 78 - 84, SIGARCH Newsletter 6,7 April 1978

Bode, A./ Händler, W.: "Rechnerarchitektur, Grundlagen und Verfahren", Arbeitsberichte des IMMD Erlangen, Bd.11,Nr.19, Erlangen Nov. 1978

Boulaye, G. G.: "Structured design for structured computer architecture", Comp. Arch News 4, 2

Bürkle, H. J./ Frick, A./ Schlier, Ch.: "High level language oriented hardware and the post-von-Neumann-era", "The 5th Annual Symposium on Computer Architecture", SIGARCH Newsletter 6, 7 April 1978, pp. 60 - 65

Burstall, R. M.: "Programm proving as hand simulation with a little induction", Information Processing 1974, Proceedings of IFIP Congress 74, North Holland, 1974, pp. 308 - 312

Constable, R. L.: "On the theory of programming logics", Proceeding of the ninth annual ACM Symposium on theory of computing, Boulder, Colorado, May 2 - 4 1977

Dahl, O.-J./ Dijkstra, E. W./ Hoare, C. A. R.: "Structured Programming", Academic Press, London 1972

Dennis, J. B.: "Computer architecture and the cost of software", Comp. Arch. News 5, 1, 1976

Deutsch, L. P.: "An interactive program verifier", Ph. D. dissertation, Dept. Comp. Sci., Univ. of Caliv., Berkeley CA., May 1973

Dijkstra, E. W.: "A discipline of programming", Prentice Hall 1976

Dijkstra, E. W.: "Guarded commands, nondeterminancy and formal derivation of programs", CACM 18, 8 August 1975

Dijkstra, E. W.: "Notes on structured programming", in: Dahl, O.-J.: Structured programming", Academic Press, London 1972

Dörner, D.: "Problemlösen als Informationsverarbeitung", Kohlhammer Standards Psychologie, Stuttgart 1976

Emden, M. H. van: "Unstructured systematic programming", Report CS-76-09, University of Waterloo, Waterloo, Ontario, Febr. 1976

Floyd, R. W.: "Assigning meanings to programs", Proc. Symp. Appl. Math., Amer. Math. Soc., vol. 19, pp. 19 - 32

Gagliardi, U. D.: "Software -related advances in computer-hardware", Comp. Arch. News 3, 2, 1974

Gries, D.: "Current ideas in programming methodology", International Summer School on Program Construction, Munich, July 26 to August 6, 1978

Grimsdale, R. L.: "Computer architectures for the interpretation of high-level languages", in: "Computer Architecture", ed. W. Händler, Springer Verlag, Berlin 1976

Guttag, J./ Horowitz, E./ Musser, D.: "Abstract data types and software validation", CACM 21, 12, Dec. 1978

Guttag, J.: "Notes on data abstraction", International Summer School on Program Construction, Munich, July 26 to August 6, 1978

Händler, W./ Hofmann, F./ Schneider, H. J.: "A general purpose array with a broad spectrum of applications", in: Computer Architecture, Workshop of the Gesellschaft für Informatik, ed. W. Händler, Informatik-Fachberichte Bd. 4, Springer Verlag 1974, pp. 311 - 335

Händler, W.: "Aspects of parallelism in computer architecture", IMACS (AICA)-GI-Symposium, "Parallel Computers - Parallel Mathematics", München, 14 - 16 March 1977

Händler, W.: "Unconventional computation by conventional equipment", Nato-Defense Research Group Seminar, Design and Evolution of Information Systems, Athen 1974

Händler, W.: "On classification schemes for computer systems in the post-von-Neumann-era", Annual Conference of Gesellschaft für Informatik, Berlin 1974

Henke, F. W./ Luckham, D. C.: "A methodology for Verifying programs", ACM SIGPLAN Notices 1975

Hirschberg, G.: "Automatisierung mit Rechnern Stand und Tendenzen", Informatik-Fachberichte 7, Springer Verlag, Berlin 1977, S. 5 - 38

Hoare, C. A. R./ Lauer, P. E.: "Consistent and complementary formal theories of the semantics of programming languages", Acta Informatica 3, 1974, pp. 135 - 153

Janov, Y. I.: "The logical schemes of algorithms", Problems of Cybernetica, vol. 1, Pergamon Press, New York 1960, pp. 82 - 140

King, J. C.: "A new approach to program testing", Programming Methodology, Lect. Notes in Comp. Sci., 23, Springer Verlag, 1974, pp. 278 - 290

King, J. C.: "A program verifier", Ph. D. dissertation, Carnegie-Mellon Univ., Pittsburgh 1969

King, J. C.: "On generating verification conditions for correctness proofs", Informatik-Fachberichte 1, ed. H.-J. Schneider/ M. Nagl, Springer Verlag 1976, pp. 253 - 267

Knuth, D. E.: "Structured programming with goto statements", Computing Surveys, Vol. 6, No. 4, Dec. 1974, pp. 261 - 301

Lampson, B. W./ Horning, J. J./ London, R. L./ Mitchell, J. G./ Popek, G. J.: "Notes on the design of Euclid", SIGPLAN Notices 12, No. 3, 1977, pp. 11 - 18

Liskov, B. H./ Zilles, S. N.: "Specification techniques for data abstractions", IEEE Transactions on Software Engineering, Vol. SE-1, No. 1, March 1975, pp. 7 - 18

Löffler, H.: "Erweiterung Boole'scher Funktionen auf den Wert 'undefiniert'", Diplom-Arbeit am IMMD (III), Erlangen, Nov. 1977

Lorenzen, P./ Schwemmer, O.: "Konstruktive Logik, Ethik und Wissenschaftstheorie", BI Hochschultaschenbücher, Bd. 700, Mannheim 1973

Luckham, D. C./ Park, D. M./ Paterson, M. S.: "On formalized computer programs", J. Comp. System Sci., 4, 1970, pp. 220 - 249

Luft, A. L.: "Ansätze für ein experimentelles Programmierlaboratorium", Vortrag beim Treffen 'Interaktive Programmierverfahren' am 7. März 1978 in Braunschweig

Luft, A. L.: "Grundlagen einer Methode für die rechnergestützte Verifikation von while-Programmen während der Programmerstellung", Arbeitsberichte des IMMD Erlangen, Bd. 8, Nr. 7, Erlangen August 1975

Manna, Z./ Waldinger, R.: "Is 'sometime' sometimes better than 'always'? Intermittent assertions in proving program correctness", CAM, 21,2, Febr. 1978

Manna, Z.: "Mathematical theory of computation", McGraw-Hill Book Company, 1974

Miller, R. E./ Cocke, J.: "Configurable computers: A new class of general purpose machines", International Symposium on Theoretical Programming, Springer Verlag, New York 1975, pp. 285 - 298

Milne, R./ Strachey, Ch.: "A theory of programming language semantics", London 1976

Müller, H.: "Diskrete Algebraische Strukturen II", Arbeitsberichte des IMMD Erlangen, Bd. 8, Nr. 2, Erlangen 1975

Müller, H.: "Programm-Schemata mit Prozeduren", Arbeitsberichte des IMMD Erlangen, Bd. 6, Nr. 6, Erlangen 1973

Oberschelp, A.: "Elementare Logik und Mengenlehre I", BI Hochschultaschenbücher, Bd. 407, Mannheim 1974

Polya, G.: "How to solve it. A new aspect of mathematical method", Princeton University Press, Princeton, New Jersey 1973

Schneider, H. J.: "Programmentwicklung als konstruktive Aufgabe", Technical Report, Lehrstuhl für Programmiersprachen, Universität Erlangen, Juni 1978

Simon, H.A.: "The sciences of artifical", Chapter 4: The architecture of complexity, The M.I.T. Press, Cambridge, Mass, 1969

Sneed, H. R.: "Software Qualitätskontrolle - Der Preis der Systemzuverlässigkeit", Online, Okt./Nov. 1977

Su, S. Y. H.: "A survey of computer hardware description languages in the USA", in: Computer, Vol. 7, Nr. 12, Dec. 1974, pp. 45 - 51

Tanenbaum, A. S.: "Implications of structured programming for machine architecture", CACM 21, 3, 1978

Waldinger, R. J./ Levitt, K. N.: "Reasoning about programs", Artif. Intell., 5, 3, 1974, pp. 235 - 316

Wand, M.: "A new incompletness result for Hoare's system", Proceedings of the eight annual ACM Symposium on theory of computing, Hershey, Pennsylvania May 3 - 5 1976

Weinberg, G. M.: "The psychology of computer programming", Van Nostrand Reinhold Company, New York 1971

Winkler, G.: "Entwurf, Implementierung und Gebrauch eines Dialogsystems zum Erstellen von Programmen unter der Steuerung von kontextfreien und kontextsensitiven Grammatiken", Dissertation, Darmstadt 1977

Wirth, N.: "Systematisches Programmieren", Teubner Verlag, Stuttgart 1972

Wulf, W. A./ London, R. L./ Shaw, M.: "An introduction to the construction and verification of ALPHARD programs", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, Dec. 1976

L e b e n s l a u f

Am 23. Januar 1944 wurde ich als viertes Kind des Flugzeugbauers Georg Luft und seiner Ehefrau Maria, geb. Frötschl, in Sorghof Kreis Tachau (Sudetenland) geboren.

Die Zwangsaussiedlung brachte meine Familie 1946 nach Trauchgau Kreis Füssen; 1953 zogen wir nach Stuttgart um.
Dort war ich von 1955 - 1961 Schüler an der Jahn-Mittelschule und danach absolvierte ich bis 1964 eine Ausbildung zum Elektro-Mechaniker.

Vom WS 1964/65 bis SS 1967 studierte ich am damaligen Ohm-Polytechnikum in Nürnberg Elektrotechnik; anschließend war ich als Entwicklungsingenieur bei der Firma Siemens AG in Erlangen tätig.

Im Herbst 1968 erwarb ich im Rahmen einer Sonderprüfung für Ingenieurschul-Absolventen in München die allgemeine Hochschulreife.

Vom WS 1968/69 bis SS 1974 studierte ich an der Universität Erlangen zunächst Elektrotechnik (bis zum Vordiplom) und nach der Einrichtung des Informatik-Studienganges Informatik.

Seit dem 15. Oktober 1974 bin ich als V. d. D. e. wiss. Assistenten am IMMD(III) der Universität Erlangen tätig.

Pinzberg, 10. Mai 1979

Alfred Luft

(Alfred Luft)

In letzter Zeit sind folgende Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung erschienen :

Band 10

NR. 1 Hoffmann, W. :
Modelle und Analysen von Teilnehmerrechensystemen, angewendet auf das Teilnehmersystems RESPOND unter CD 3300 MASTER.
(Februar 1977)

NR. 2 Körber, K., Werzinger, G. :
Das Modell SIM-LAB.
Simulation eines klinisch-chemischen Laboratoriums.
(März 1977)

NR. 3 Altmann, W., Weber, D. :
Programmiermethodik.
(März 1977)

NR. 4 Maehle, E. :
Fehlertolerante Rechnerstrukturen.
(April 1977)

NR. 5 Beth, T. :
Kombinatorik: Theorie der Vertretersysteme.
(April 1977)

NR. 6 Mertens, P. :
Einige Hinweise zur technischen Gestaltung von Bildschirmdialogen.
(Mai 1977)

NR. 7 Jäpel, D. :
Ein Begriffssystem zur Formulierung von Schutzaspekten und seine Anwendung auf Programmsysteme.
(Juni 1977)

NR. 8 Brendel, W., Grötsch, E., Nagl, M., Schneider, H.-J. :
Graphersetzungssysteme, Beiträge zur Theorie, Anwendung und Implementierung.
(Juni 1977)

NR. 9 Rüdel, W. :
Darstellung und Vergleich abstrakter Modelle zur Beschreibung geschützter Systeme.
(Juli 1977)

NR. 10 Bolch, G., Gall, R., Grassl, M., Schmidt, B. :
Modelle für Trommelspeicher.
(Juli 1977)

NR. 11 v. Eichhorn, K. :
Steiner Systeme, Codes und affine Räume.
(Juli 1977)

Band 10 (Forts.)
==========

NR. 12 Göttler, H. :
Zweistufige Graphmanipulationssysteme für die Semantik von Programmiersprachen.
(Juli 1977)

NR. 13 Keramidis, S. :
Ein allgemeines formales Modell für Betriebssysteme und dessen Implementierung.
(September 1977)

NR. 14 Arnold, F., Benker, L., Grütz, M. :
Die Fast Fourier Transformation und Möglichkeiten ihrer Implementierung auf EGPA.
(November 1977)

NR. 15 Gardill, R., Händler, W., Hessling, H., Klar, R., Spies, P. P. :
ERES
Eine nichtprozedurale Rechnerentwurfssprache mit präziser Zeitbeschreibung.
(November 1977)

NR. 16 Wichmann, H. :
Entwicklung und Bewertung von Algorithmen für ein mikroprogrammiertes Parallel- und Assoziativkonzept auf Mikroprogramm- und Benutzerebene.
(Dezember 1977)

NR. 17 Brügmann, H. W. :
EBIDIS - Strukturierte Bildprogrammierung im Dialog.
(Dezember 1977)

NR. 18 Jahresbericht 1977 der Informatik.
(März 1978)

Band 11
=======

NR. 1 Körber, K., Werzinger, G. :
Das Modell SIM-LAB.
Beschreibung des Grundmodells.
(Februar 1978)

NR. 2 Körber, K., Werzinger, G. :
Das Modell SIM-LAB.
Beschreibung der Modellerweiterungen und der Strategien.
(Februar 1978)

NR. 3 Körber, K., Werzinger, G. :
Das Modell SIM-LAB.
Listing der Quellprogramme.
(Februar 1978)

NR. 4 Beth, Th. :
On Resolutions of Steiner Systems.
(März 1978)

NR. 5 Leeb, K. :
Salami-Taktik beim Quader-Packen.
Riess, W. :
Zwei Optimierungsprobleme auf Ordnungen.
(April 1978)

NR. 6 Händler, W., Henning, W., Kleinöder, W., Kneissl, J., Volkert, J. :
EPIK - Erlanger Projekt Interaktive Kartographie.
(April 1978)

NR. 7 Schreiber, H. :
Hardware-Messung und Analyse des Ablaufgeschehens in Rechnerkernen.
(April 1978)

NR. 8 Weber, D. :
Datengraphen und deren Transformation :
Ein Konzept zur Spezifikation von Datentypen.
(Juni 1978)

NR. 9 Musielak, H., Schmidt, B., Stössel, M. :
Vergleich von Simulationssprachen :
GPSS-FORTRAN - GPSS - GASP IV.
(Juli 1978)

NR. 10 Solymosi, A. :
Synthese von analysierenden Automaten auf Grund von formalen Grammatiken.
(Juli 1978)

NR. 11 Meyer, K., Schaller, K., Schmidt, B. :
SIM-QUEUE.
Beschreibung eines Simulationsmodells für ein komplexes Warteschlangensystem, das mit Hilfe von Eingabedaten zusammengestellt werden kann.
(Juli 1978)

Band 11 (Forts.)
==========

NR. 12 Bodendorf, F., Eckardt, T., Mertens, P., Schrammel, D. :
Benutzung und Beurteilung von Kleinrechnersystemen in einer Hochschule - Ein Beitrag zur Dezentralisierungsdiskussion.
(August 1978)

NR. 13 Rabinovitz, M. :
An Approach to a Reconfigurable Multiprocessor System.
Bode, A., Händler, W. :
Classification d´architectures parallèles: Introduction de la notation ECS et application au projet EGPA.
Sigmund, W. :
Parallel Machine Functions in Program Transformation and Execution.
(September 1978)

NR. 14 Beth, Th., Strehl, V. :
Materialien zur Codierungstheorie.
(September 1978)

NR. 15 Wendler, K. :
Betriebssystemaspekte in hierarchisch modularen Polyprozessorsystemen - Modellierungsansätze und Koordinierungsmechanismen.
(Oktober 1978)

NR. 16 Altmann, W. :
Beschreibung von Programmoduln zum Entwurf zuverlässiger Softwaresysteme.
(Oktober 1978)

NR. 17 Hoffmann, W. :
Warteschlangenmodelle für die Parallelverarbeitung.
(November 1978)

NR. 18 Stöth, G. :
Höhere Programmiersprachen für Kleinrechner : Anforderungen, Entwicklungskriterien, Compileraspekte.
(November 1978)

NR. 19 Bode, A., Händler, W. :
Rechnerarchitektur - Grundlagen und Verfahren.
(November 1978)

NR. 20 Kneissl, F. :
Grundsoftware für die Textverarbeitung.
(Dezember 1978)

NR. 21 Leidel, W. :
Synchronisationsprobleme in Automatennetzen.
(Dezember 1978)

Band 11 (Forts.)
==========

NR. 22 Bodendorf, F., Renninger, W., Schaller, K. :
Zwei neue Methodenbanksysteme für statistische Anwendungen.
(Dezember 1978)

NR. 23 Jahresbericht 1978 der Informatik.
(März 1979)

Band 12
=======

NR. 1 Elzer, P. :
Strukturierte Beschreibung von Prozessystemen.
(Februar 1979)

NR. 2 Wunderatsch, H. :
Deterministisches Zuordnen in heterogenen
Prozessorsystemen mit Pseudo-Boole Methoden.
(Juni 1979)

NR. 3 Bunke, H. :
Sequentielle und parallele programmierte
Graph-Grammatiken.
(Juni 1979)

—

Offsetdruck · Fotodruck · Hogl · Hauptstraße 109 · 8520 Erlangen · Tel.09131/22990